ENGINEERING

UNIVERSITY OF SOUTHERN CALIFORNIA

IMAGE UNDERSTANDING RESEARCH

Semiannual Technical Report

Ramakant Nevatia
Alexander A. Sawchuk
Principal Investigators
(213) 743-5506

Covering Research Activity During the Period
1 April 1980 through 30 September 1980

30 September 1980

Image Processing Institute
University of Southern California
University Park
Los Angeles, California 90007

Sponsored by

Contract No. F-33615-80-C-1080
DARPA Order No. 3119

IPI

IMAGE PROCESSING INSTITUTE

LEVEL II

(12)

USCIPI Report 990

(6) # IMAGE UNDERSTANDING RESEARCH

(9) Semiannual Technical Report, 1 Apr - 30 Sep 80,

Covering Research Activity During the Period
1 April 1980 through 30 September 1980

(10) Ramakant/Nevatia
Alexander A./Sawchuk
Principal Investigators
(213) 743-5506

(14) USCIPI-990

(12) 210

Image Processing Institute
University of Southern California
University Park
Los Angeles, California 90007

(11) 30 September 1980

This research was supported by the Defense Advanced Research Projects Agency and was monitored by the Wright Patterson Air Force Base under Contract (15) F33615-80-C-1080, DARPA Order ~~No.~~-3119.

391141

JOB

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER USCIPI Report 990	2. GOVT ACCESSION NO. AD-A094C06	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) IMAGE UNDERSTANDING RESEARCH		5. TYPE OF REPORT & PERIOD COVERED Semiannual Tech. Report 1 Apr. 80 - 30 Sept. 80
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (Principal Investigators) Ramakant Nevatia Alexander A. Sawchuk		8. CONTRACT OR GRANT NUMBER(s) F-33615-80-C-1080
9. PERFORMING ORGANIZATION NAME AND ADDRESS Image Processing Institute University of Southern California University Park, Los Angeles, Ca. 90007		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS DARPA Order No. 3119
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209		12. REPORT DATE September 30, 1980
		13. NUMBER OF PAGES 213
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Wright Patterson Air Force Base U.S. Air Force Air Force Avionics Laboratory Wright Patterson AFB, Ohio 45433		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for release: distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Key Words: Digital Image Processing, Image Restoration, Scene Analysis, Image Understanding, Edge Detection, Image Segmentation, Image Matching, Texture Analysis, VLSI Processors.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This technical report summarizes the image understanding and VLSI system research activities performed by the USC Image Processing Institute and the Hughes Research Laboratories during the period of 1 April 1980 through 30 September 1980 under contract number F33615-80-C-1080 with the Defense Advanced Research Projects Agency, Information Processing Techniques Office. This contract is monitored by the Air Force Wright

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(*When Data Entered*)

Aeronautical Laboratories, Wright-Patterson Air Force Base, Dayton, Ohio.

—The purpose of this research program is to develop techniques and systems for understanding images, particularly for mapping applications. The research activity includes low level image analysis and feature extraction, statistical and structural texture analysis, symbolic image representations, and image to map correspondence using relational structures. Additional activity is concerned with VLSI architectures for implementation of image analysis and image understanding operations.

UNCLASSIFIED

SECURITY CLASSIFICATION OF TH AGE(*When Data En*

ABSTRACT

This technical report summarizes the image understanding and VLSI system research activities performed by the USC Image Processing Institute and the Hughes Research Laboratories during the period of 1 April 1980 through 30 September 1980 under contract number F33615-80-C-1080 with the Defense Advanced Research Projects Agency, Information Processing Techniques Office. This contract is monitored by the Air Force Wright Aeronautical Laboratories, Wright-Patterson Air Force Base, Dayton, Ohio.

The purpose of this research program is to develop techniques and systems for understanding images, particularly for mapping applications. The research activity includes low level image analysis and feature extraction, statistical and structural texture analysis, symbolic image representations, and image to map correspondence using relational structures. Additional activity is concerned with VLSI architectures for implementation of image analysis and image understanding operations.

TABLE OF CONTENTS

1. RESEARCH OVERVIEW

This report describes the results of our research under contract F-33615-80-C-1080. This report also contains results of research only partially supported by this contract. Our research includes work at many levels of an Image Understanding system, including low level feature extraction, symbolic descriptions and image matching. We have also been working with Hughes Research Laboratories on hardware implementation of IU algorithms, using VLSI technology. The research results are summarized below.

Image Matching

Matching of an image to a symbolic map, or a symbolic description of another image, is central for the tasks of map updating and change detection. In the past we have described results using aerial images and a relaxation matching algorithm developed by Faugeras and Price. Some new results using this algorithm are described in section 2.1.

Texture Analysis and Synthesis

Presence of texture is a dominant cause of difficulty in the analysis of aerial images. We have continued development of several texture analysis and synthesis techniques.

In previous work, we developed techniques of structural texture analysis by examining repetition patterns of micro edges. These techniques obtained isolated 2-dimensional texture primitives and the period of repetition, if any. For complex textures, the texture primitives of different kinds are also related in certain ways, and in some cases, a "super-primitive" is repeated regularly. Section 2.2 describes techniques for computing relations between texture primitives.

A technique of texture analysis by Singular Value Decomposition (SVD) is described in section 2.3. This section describes new results of research supported by earlier contracts, and this work is not being actively pursued now.

Results of texture synthesis using stochastic models are given in sections 2.4 and 2.5. Synthesized textures have very similar appearance to the corresponding natural textures.

Shape Description and Matching

Shape of structural objects can be described by relationships of their components. A technique of connecting fragments of runway descriptions into complete runways is given in section 2.6.

Sections 2.7 and 2.8 describe use of hierarchical gradient relaxation techniques for matching of shapes including scenes with occlusion.

Other IU Projects

Sections 2.9 and 2.10 present new results of previously reported projects on motion detection and range data processing.

Hardware Implementation

In continuing work with Hughes Research Laboratories, Malibu, California, we are investigating the use of VLSI technology for hardware implementation of IU algorithms. We have chosen to investigate the following algorithms initially:

i) Nevatia-Babu Line Finder
ii) Ohlander Region Segmentor
iii) Laws Texture Analysis System

The choice of the above three algorithms was based on their computation intensive nature, their use for a broad range of problems and experience with a large number of images for the first two. Also these algorithms are largely local and hence easier to implement in VLSI hardware, where reducing interconnections is important. Further, the three algorithms have common kernels, such as convolution, but also require different subsequent processing.

Detailed logic designs resulting in estimates of gates and the number of chips required to implement the line finder and texture system have been completed and reported in Section 3. Also, we have identified a common kernel for these algorithms and designed a modular programmable digital processing element RADIUS. It is based on novel concepts of residue arithmetic and can perform a variety of local area processing operations including convolution with very high circuit function density.

2. IMAGE UNDERSTANDING PROJECTS

2.1 More Results on Application of Relaxation Matching

K.E. Price

The relaxation algorithm was described in detail in the previous semi-annual report so it will not be discussed here. In this one we will only present some additional results. In one of the images of the previous report there were 3 clusters of tanks with 14, 11 and 6 objects in each of the groups. Fig. 1 gives the results for the matching of these 3 groups. The segmentation of what should have been T10 and T14 in the upper group was imperfect. T14 was matched to a fragment but T10 was not. Fig. 2 illustrates how the most likely match changes through the micro and macro iterations described previously. Initially, the tanks are assigned based only on feature values, so that all are assigned to the image segment closest to the model tank size. After a few micro iterations, the relations become more important and some assignments become stronger. By the second macro iteration, the relations contribute enough to the initial assignments that they are mostly correct, which is not the case of potential assignments at the end of the first macro iteration.

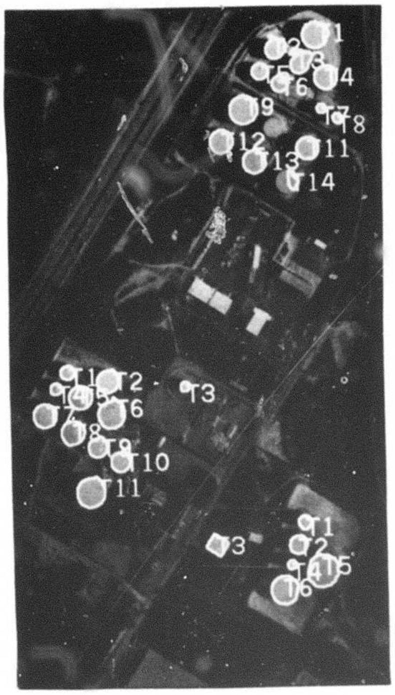
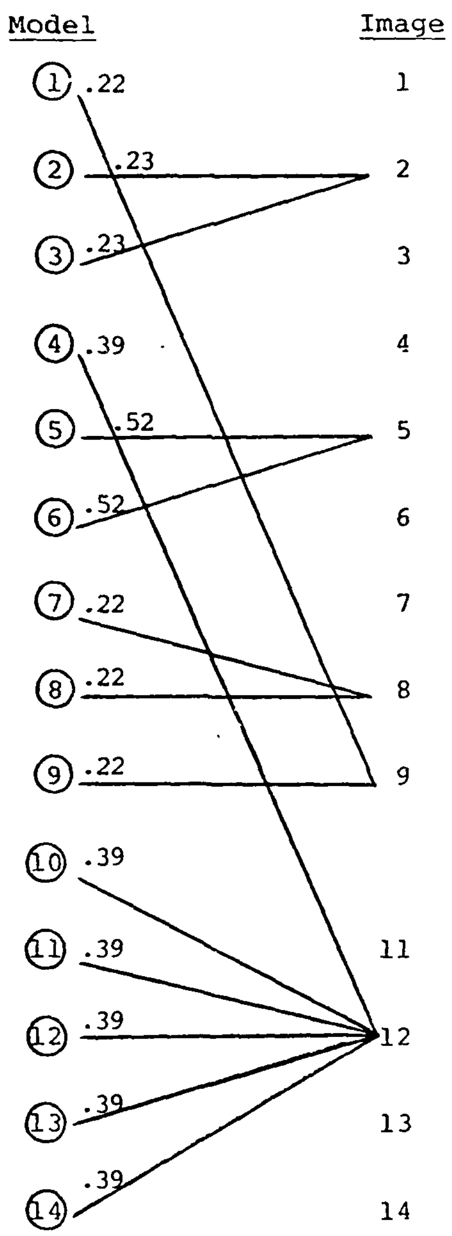
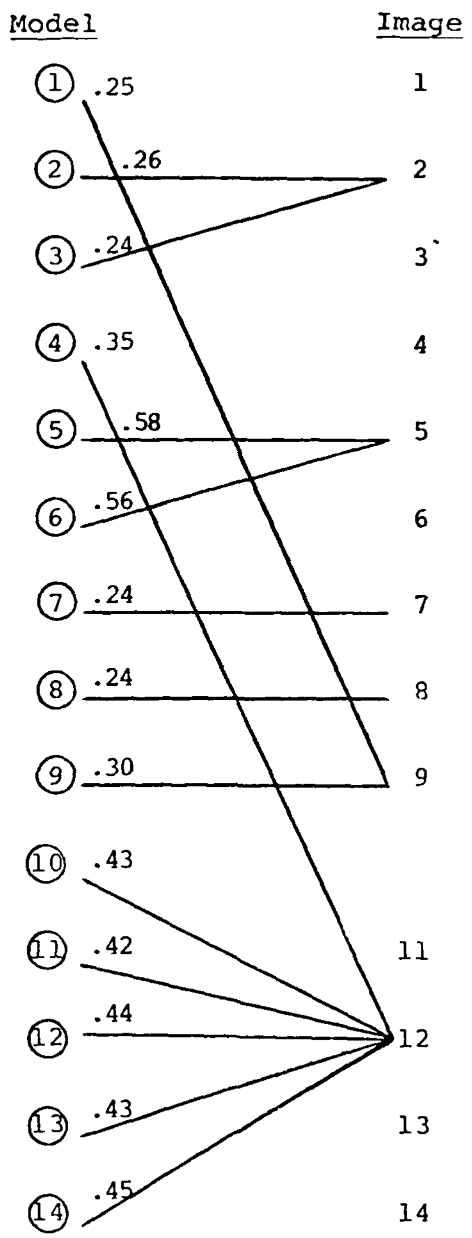


Fig. 1. Results for matching on 3 clusters of tanks.

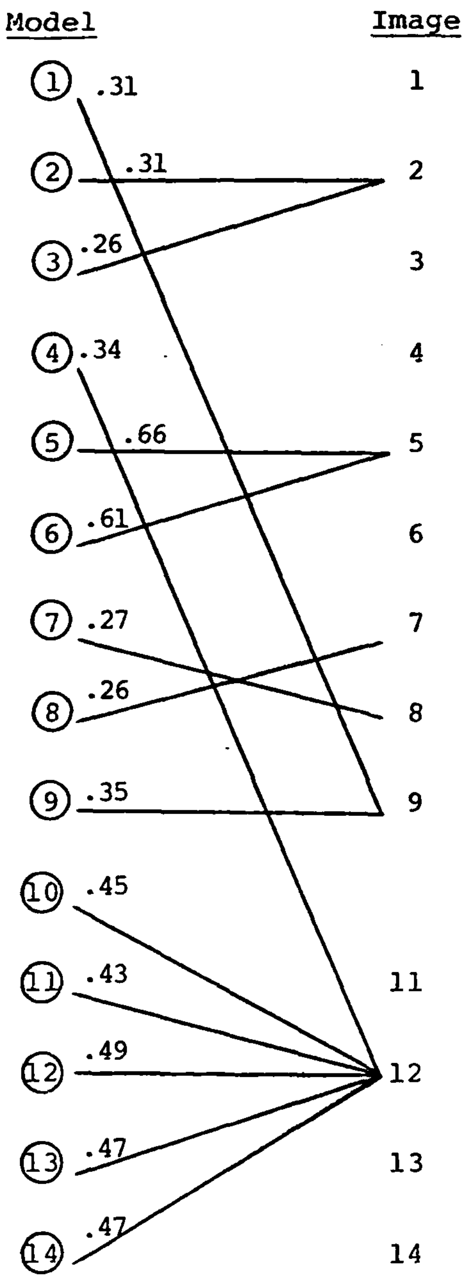


Initial assignments

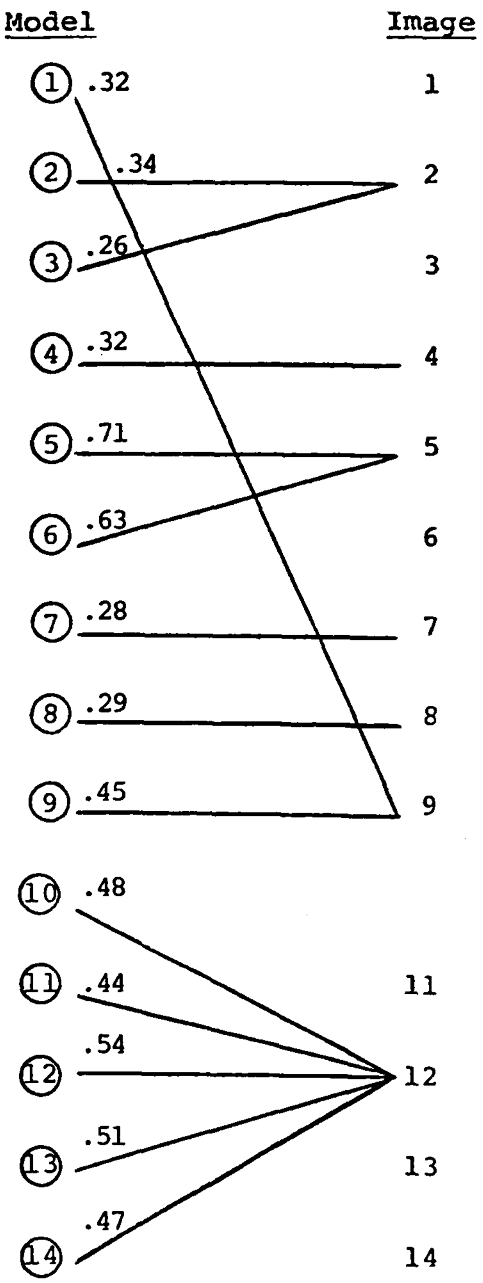


Micro iteration 1

Fig. 2 Sequence showing the changing most likely assignments for all objects in the top cluster of tanks (Fig. 7). The initial assignments are given with the results for each micro iteration. At the end of the macro iteration several units are labeled (indicated by *), and these labels are carried to the following iterations (indicated by **). In the later macro iterations the assignments do not change so that the results are indicated with the initial likelihood values.

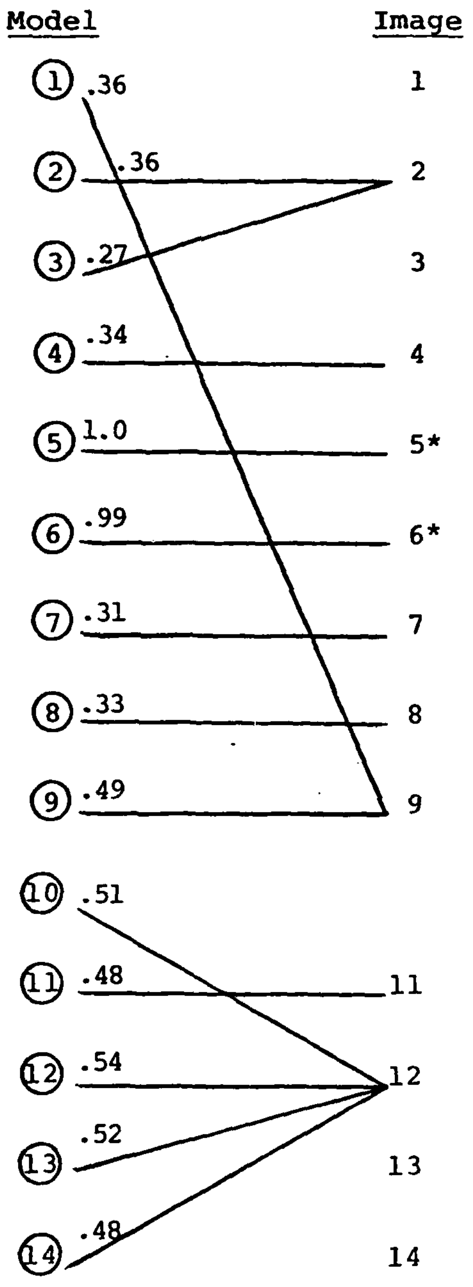


Micro iteration 2

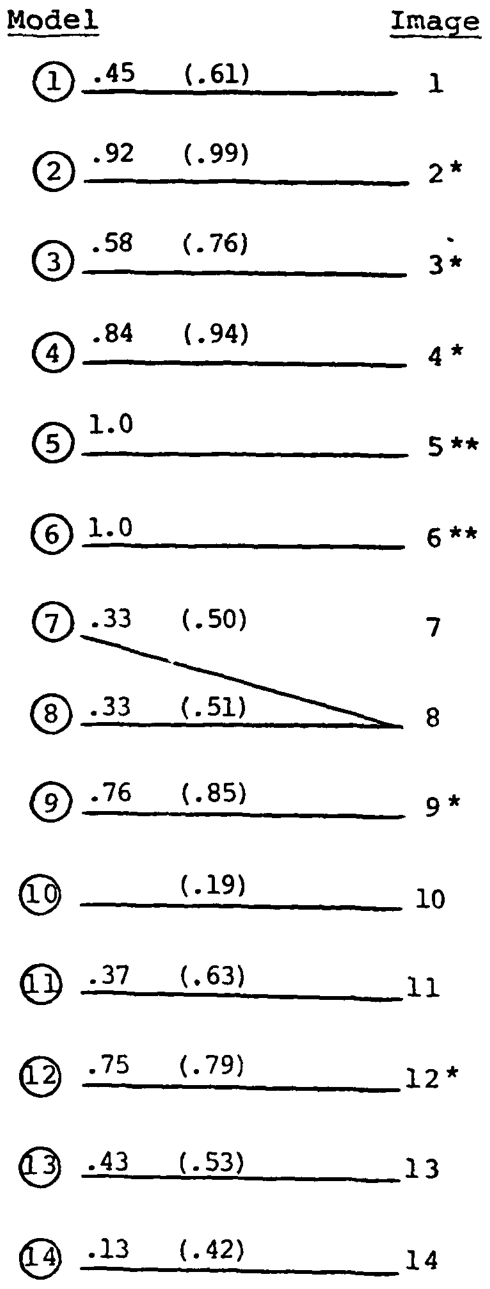


Micro iteration 3

Fig. 2 continued.

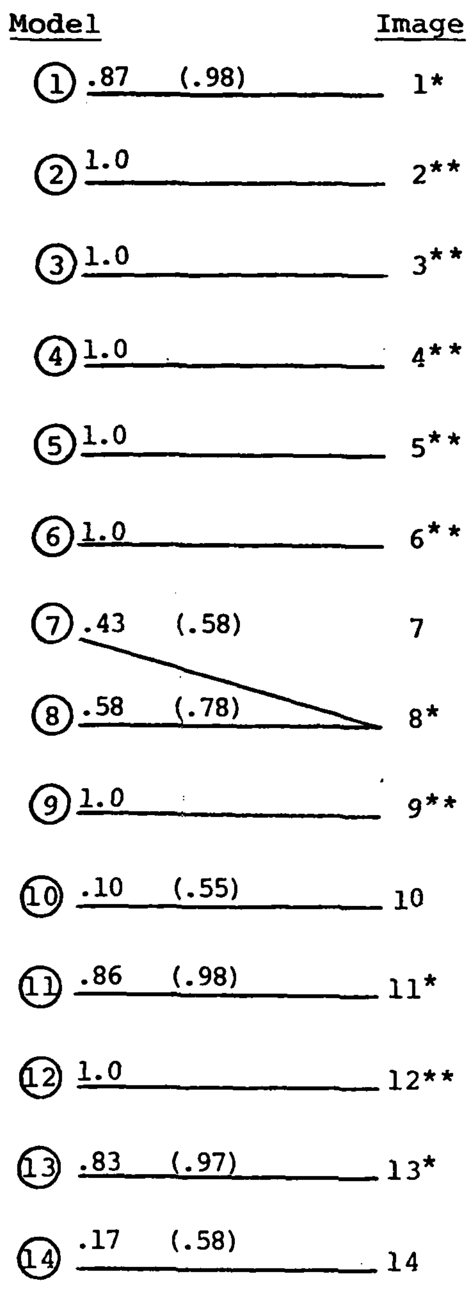


Micro iteration 4
End of macro iteration 1

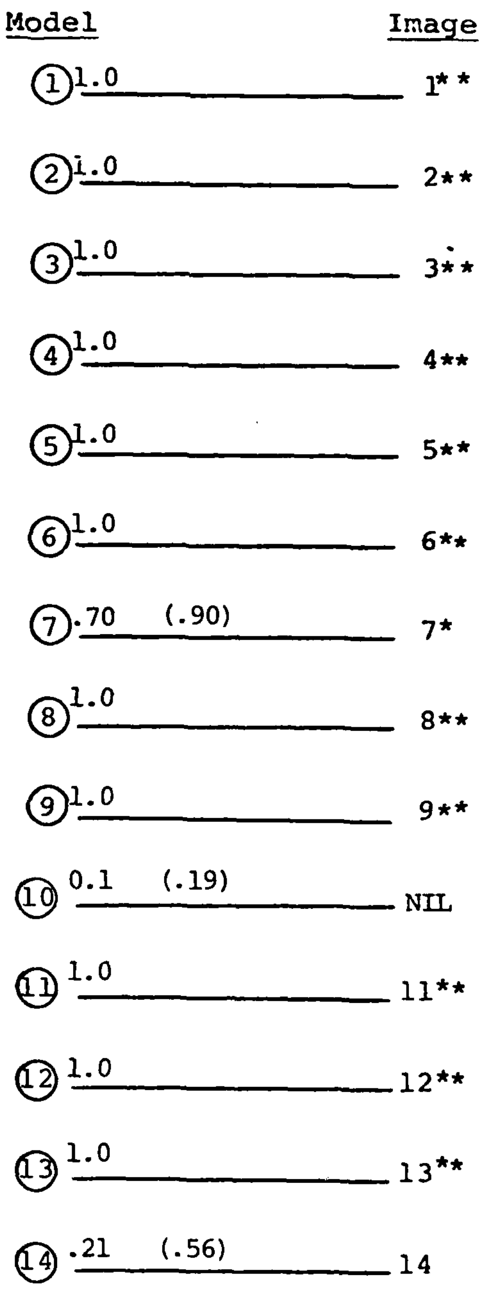


Macro iteration 2
Initial assignments
and micro iteration 5
- in parenthesis.

Fig. 2 continued.



Macro iteration 3
Initial assignments
and micro iteration 6
6 - in parenthesis.



Macro iteration 4
Initial assignments
and micro iteration 7
7 - in parenthesis.

Fig. 2 continued.

2.2 Determining Spatial Relationships Between Texture Primitives in Homogeneous Regular Textures

F. Vilnrotter*, R. Nevatia, and K.E. Price

Introduction

In previous reports, we have described programs used to generate descriptions of natural textures [1-2] and extract and describe texture primitives [3]. (These reports also contain a discussion of other work and a list of related references). Short descriptions of these programs may also be found in [4-5].

In the first part of the program edge repetition arrays are produced using the edge and direction images corresponding to the original image (an edge repetition array is the binary case of a gray level co-occurrence matrix). These arrays are calculated for 6 directions (0, 30, 60, 90, 120, and 150 degrees), for both dark and light intensity objects, at distances within a range specified by the user. A comprehensive discussion of edge repetition arrays is given in [1].

In the second part of the program the edge repetition arrays are analyzed to determine whether there are predominant element sizes in any of the 6 scan directions and if so whether these elements occur at regular intervals within the image. The details of this analysis are presented in [2] and will not be repeated here.

In the third part of the program groups of texture primitives are extracted and described, using the texture descriptions generated in part 2. Primitives are represented as masks. These masks are then

*Felicia Vilnrotter is supported by a Hughes Aircraft Company Doctoral Fellowship.

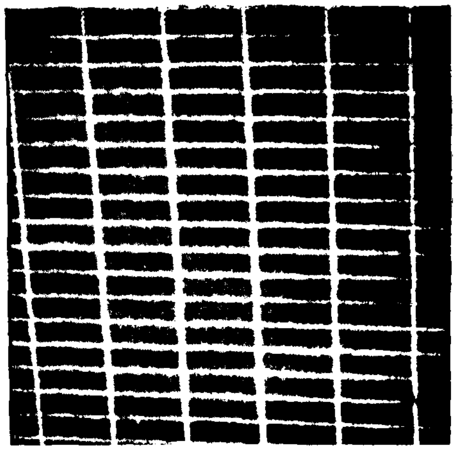
individually analyzed to determine the characteristics (average size, intensity, etc.) of each primitive group. The details of the primitive extraction and description processes are given in [3].

To summarize: analysis of edge repetition arrays provided one dimensional texture descriptions. These descriptions are the starting point for a two dimensional texture primitive search. The products of this search are a set of texture primitive masks and descriptions.

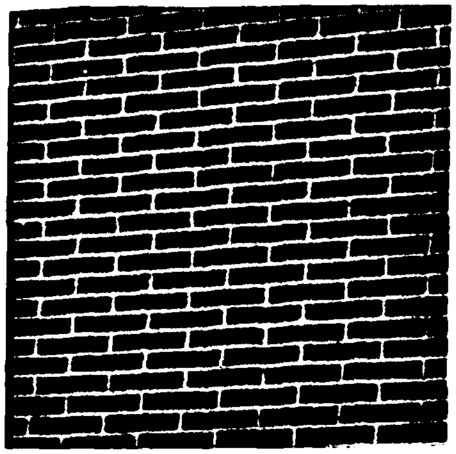
Up to this point the only descriptive information we have relating to the arrangement of the texture primitives is the element spacing information produced in part 2. However, this information pertains to only a single scan direction. Therefore, we can tell only if a certain group of primitives in periodic in one direction and if so, what period is exhibited. When no element spacing can be found for any of the elements within a texture, the texture pattern is assumed to be random.

In the event that we are considering a non-random texture pattern the spacing information described above is not sufficient to characterize the particular placement rules within the texture.

Consider the 2 brick patterns in Figure 1 (a and b). These patterns are similar in that each contains light and dark textural elements which are rectangular and are arranged in a regular pattern. The arrangement of the bricks within the patterns is different, but no evidence of this difference is given in their primitive descriptions (see Figure 2 a and b). Both sets of bricks were detected by scanning vertically. Both exhibit an element spacing in this direction. However, a definitive description of element arrangement is lacking. This placement information is contained in the primitive masks and composite images (see figs. 3 and 4), and is the object of analysis in the next program section.



(a) Brick Pattern 1 Subwindow.



(b) Brick Pattern 2 Subwindow.

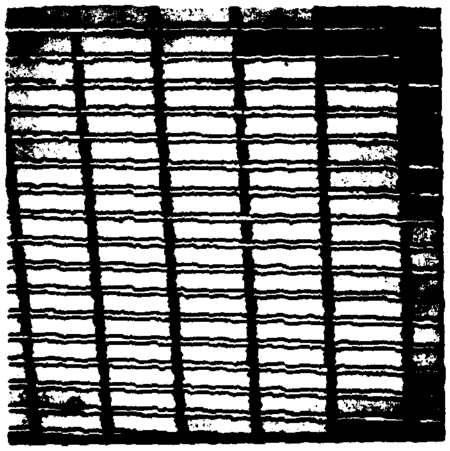
Figure 1.

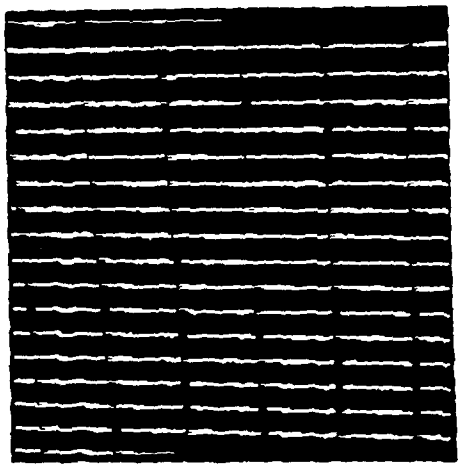
```
PRIMITIVE ANALYSIS FOR BRICK 1


RELATIVE INTENSITY IS  LIGHT   DIRECTION IS VERTICAL

NUMBER OF SAMPLES:  87      PRIMITIVE NUMBER IS:  1

  AVERAGE PRIMITIVE DIMENSIONS ARE: (2.00 AND  52.92)

  AVERAGE PRIMITIVE SIZE IN PIXELS IS:  (105.39)

  AVERAGE PRIMITIVE INTENSITY IS:   (120.80)

  PRIMITIVES REPEAT AT ELEMENT SPACING: (15.00) IN ABOVE MENTIONED DIRECTION


RELATIVE INTENSITY IS    DARK   DIRECTION IS VERTICAL

NUMBER OF SAMPLES:  106     PRIMITIVE NUMBER IS:  2

  AVERAGE PRIMITIVE DIMENSIONS ARE: (10.00 AND 35.34)

  AVERAGE PRIMITIVE SIZE IN PIXELS IS:  (353.14)

  AVERAGE PRIMITIVE INTENSITY IS:   (100.59)

  PRIMITIVES REPEAT AT ELEMENT SPACING: (15.00) IN ABOVE MENTIONED DIRECTION
```

Figure 2a. Brick pattern 1 primitive texture element description.

```
PRIMITIVE ANALYSIS FOR BRICK 2


RELATIVE INTENSITY IS  LIGHT    DIRECTION IS VERTICAL

NUMBER OF SAMPLES: 39       PRIMITIVE NUMBER IS:  1

  AVERAGE PRIMITIVE DIMENSIONS ARE: (2.00 AND  145.26)

  AVERAGE PRIMITIVE SIZE IN PIXELS IS:  (289.87)

  AVERAGE PRIMITIVE INTENSITY IS:  (175.44)

  PRIMITIVES REPEAT AT ELEMENT SPACING:  (12.00) IN ABOVE MENTIONED DIRECTION


RELATIVE INTENSITY IS   DARK    DIRECTION IS VERTICAL

NUMBER OF SAMPLES:  122      PRIMITIVE NUMBER IS:  2

  AVERAGE PRIMITIVE DIMENSIONS ARE:  (8.00 AND 40.93)

  AVERAGE PRIMITIVE SIZE IN PIXELS IS:  (326.57)

  AVERAGE PRIMITIVE INTENSITY IS:   (134.25)

  PRIMITIVES REPEAT AT ELEMENT SPACING:  (12.00) IN ABOVE MENTIONED DIRECTION
```

Figure 2b. Brick pattern 2 primitive texture element description.


(a) Brick pattern 1 composite primitives.


(b) Light brick pattern 1 primitive found in vertical scan.

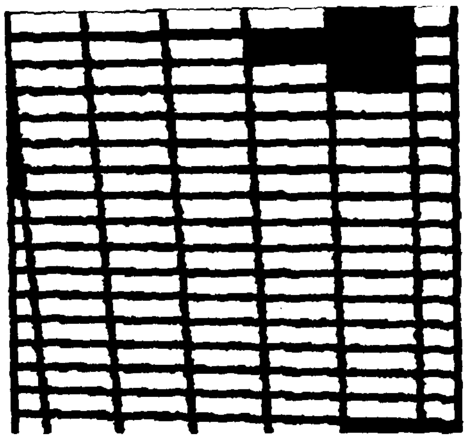
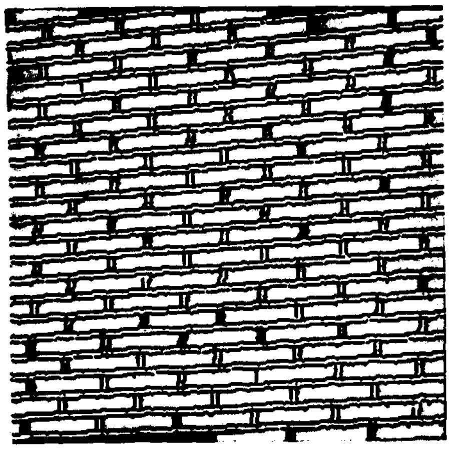
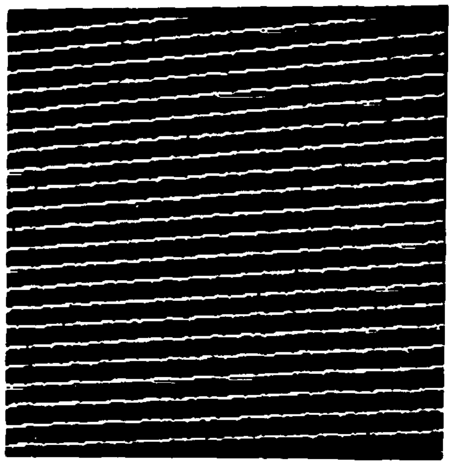

(c) Dark brick pattern 1 primitive found in vertical scan.

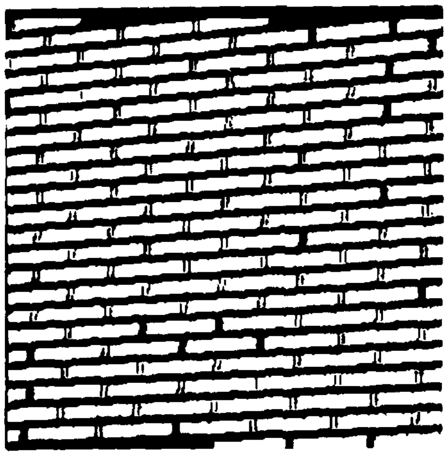
Figure 3.



(a) Brick pattern 2 composite primitives.



(b) Light brick pattern 2 primitive found in vertical scan.



(c) Dark brick pattern 2 primitive found in vertical scan.

Figure 4

Methodology

Individual primitive masks provide the locations of all the primitives of a particular type within an image. Determining the predominant placement rules exhibited by these elements can be divided into a number of independent subtasks:

1) It must be determined whether 2 primitive masks represent the same textural elements. This situation arises when a textural element is detected in more than one scan direction.

Implementation - the 2 masks under consideration are ANDED to create a new mask. If the resulting mask contains more than a given percentage of the non-zero points of either of the original masks the 2 masks are combined and their descriptions are linked. All pairs of primitive masks are tested and the result is a new set of masks and a reduced primitive list.

2) Inter-primitive distances will be measured along the lines connecting primitive centroids. Therefore, all primitive centroids must be known.

Implementation - each primitive in each primitive mask is catalogued along with coordinates of its center of mass and its primitive type. An image containing the index into this list in the locations corresponding to the non-zero entries of the mask for that particular textural primitive is created for future reference during the matching process.

3) From the primitive centroids the image will be scanned in 12 directions in the search for predominant spatial relationships.

Implementation - the scan originating at the center of mass of a primitive proceeds in each of 12 directions. When a different primitive is encountered its type, say A, and centroid location are noted. The angle and distance between the 2 centroids is also noted and a 1 is added to the bin indexed by that primitive type (A), the angle (within 10 degrees), and the distance (within 3 pixels). All of the results of the scans originating from the same primitive type are considered together. They are stored in a temporary matrix for further processing. Only the spatial relationships which occur most frequently are part of the final results.

4) The raw counts of primitive matches must be normalized and thresholded so that only the matches occurring most often will be considered.

Implementation - All matches generated by starting from the centroids of a particular type of primitive, say A, are stored in a temporary matrix. The following steps are then taken:

(i) Each entry is normalized as follows. Let M=# Matches from primitive type A to primitive type B at angle θ and distance D. Then

$$\text{Normalized (M)} = \frac{100 \times M}{\#(\text{Prims. of Type A})\frac{(\#\text{Rows}-D \sin\theta)(\#\text{Cols}-D \cos\theta)}{(\#\text{Rows} \times \#\text{Cols})}}$$

(ii) All matrix entries greater than or equal to a certain threshold, (THR1*Maximum Matrix Value), are replaced by the sum of 9 entries. These 9 entries are those whose degree and distance indices are not different from the original entry's indices by more than 1. This should insure that any large volume peak which is spread out will not be passed over in the next thresholding process.

(iii) A new matrix maximum, NEWMAX, is calculated. Any entry greater than or equal to a certain threshold, (THR2*NEWMAX) which is also a local maximum is added to the set of final results.

These final results are a set of spatial relationships of the form:

PRIM1 = A   PRIM2 = B   Angle = θ DISTANCE = D   PERCENTAGE = P

This relationship states that the centroid of a primitive of type A is separated from the centroid of a primitive of type B by a distance of D pixels at angle θ P percent of the time that a primitive of type A is encountered.

Results

The spatial relationships generated for the 2 brick texture patterns are tabulated in figures 5a and 6a. Please note that angles are found to the nearest 10 degrees, distances to the nearest 3 pixels. The salient spatial characteristics of Brick Pattern 1 (BP1) are captured in the spatial relationships generated for this pattern (see figure 5a). Now consider Brick Pattern 2 (BP2). The primitive placement results or spatial relationships given in figure 6a are not as strong as the results for the first brick pattern. All of the spatial relationship percentages for BP1 are above 40%, while there are none above 40% for BP2. Also, there are many more relationships listed for the second pattern.

A comparison of the brick primitives of BP1 (figure 3c) with the brick primitives of BP2 (figure 4c) will help to provide some of the reasons for this difference. Many of the brick primitives in figure 4c are connected to neighboring bricks or mortar due to missing edges. When 2 bricks are connected to form one primitive its centroid will appear directly in line with the centroids of the bricks above and below it. If this happens enough times the brick primitive will appear to be related to itself at angles of 90 and -90 degrees. This relationship does appear in figure 6a with a relatively high percentage or relational occurrence rate. Along with this effect there is the lowering of the percentages of the actual (or correct) primitive relationships making thresholding less effective. Intermittently the vertical strips of mortar separating neighboring

Brick1 Primitive Placement Results

PRIM1	PRIM2	ANGLE[1] (DEGREES)	DISTANCE (PIXELS)	TOTAL (%)
1	1	-90	15	52.90
1	1	90	15	52.90
1	2	-90	6	61.36*
1	2	90	6	52.25*
2	1	-90	6	42.38
2	1	90	6	50.53
2	2	-180	45	64.09*
2	2	-90	15	94.55*
2	2	0	45	63.12*
2	2	90	15	94.55*

*These are used in part b.

[1]Negative angles are counter-clockwise; positive angles are clockwise.

Figure 5a. Brick pattern 1 interprimitive angle in degrees, distance in pixels and frequency of occurrence.

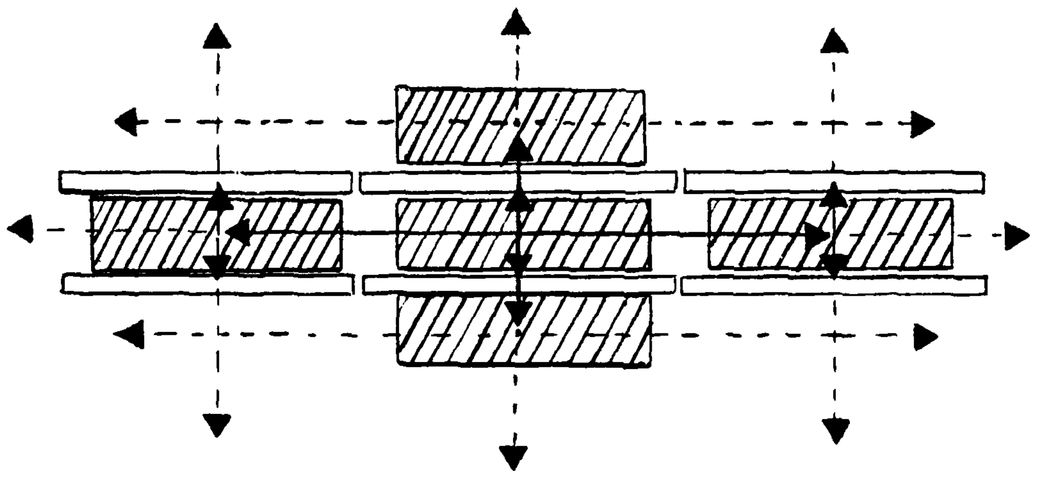


Figure 5b. Brick pattern 1 texture reconstruction using information in a.

Brick2 Primitive Placement Results

PRIM1	PRIM2	ANGLE[1] (DEGREES)	DISTANCE (PIXELS)	TOTAL (%)
1	1	-90	12	18.28*
1	1	-50	15	13.11
1	1	-40	21	13.20
1	1	0	132	15.48*
1	1	90	12	18.27*
1	1	130	15	13.14
1	1	140	21	13.26
1	2	-170	18	18.48
1	2	-170	24	13.34
1	2	-170	30	16.17
1	2	-30	12	23.48*
1	2	0	48	13.68
1	2	0	72	14.14
1	2	10	24	18.52
1	2	10	33	18.76
1	2	80	6	20.70
1	2	100	6	18.11
2	2	-160	24	30.56*
2	2	-90	12	22.24
2	2	-30	27	27.03
2	2	0	24	19.93
2	2	0	45	33.00*
2	2	20	24	30.38*
2	2	90	12	23.10
2	2	150	27	26.32
2	2	180	24	20.08
2	2	180	45	32.57*

*These are used in part b.

[1]Negative angles are counter-clockwise; positive angles are clockwise.

Figure 6a. Brick pattern 2 interprimitive angle in degrees, distance in pixels and frequency of occurrence.

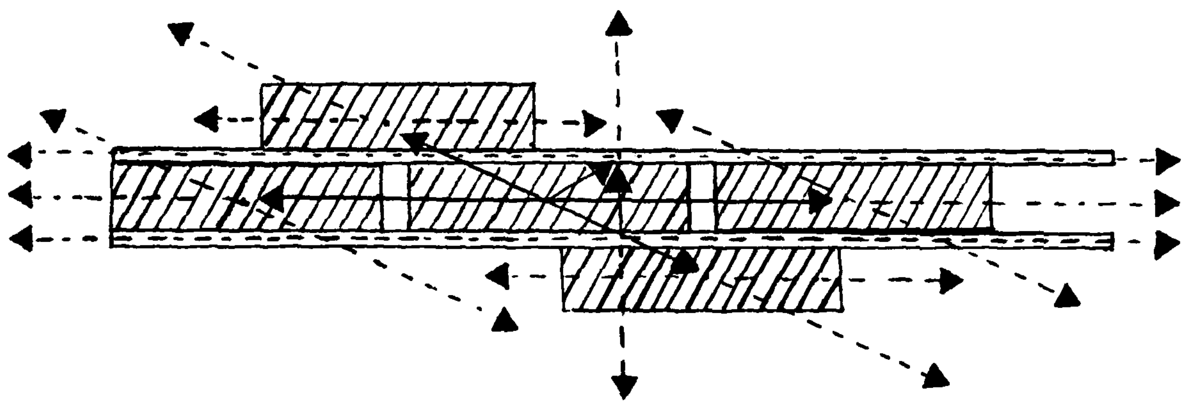


Figure 6b. Brick pattern 2 texture reconstruction using information in a.

bricks appear as separate primitives. This is responsible for the occurrence of the horizontal brick relationships for BP2 at distance 24 with percentage 19.93 for 0 degrees and 20.08 at 180 degrees.

Type 1 primitives, the horizontal mortar strips, are also a problem. (Compare figure 3b for BP1 with figure 4b for BP2). In BP2 the mortar strips are connected to form primitives which are too long to generate meaningful spatial relationships and too short to be considered as part of the background.

In spite of these problems a systematic search of the information in figure 6a can lead to a reasonable structural description of the texture pattern. First try to establish the repetition pattern of each set of primitives along 2 non-colinear lines. Hopefully this can be done for at least 1 primitive set. Starting with the strongest grid pattern established try to relate all other primitives to the primitives composing this grid.

In BP1 the only primitive with spatial relations in 2 non-colinear directions is primitive 2, the brick primitive. It forms a rectangular grid pattern since its spatial relations occur at angles -180, -90, 0, and 90 degrees. The mortar primitive, primitive 1, is related to the brick primitives by vertical spatial relationships at distance 6. Using the primitive analysis information in figure 2a the textural pattern can be filled in to produce figure 5b.

In BP2 the primitive exhibiting the strongest spatial relationships is the brick primitive, primitive 2. Its strongest spatial relationships occur at 0 and 180 degrees at a distance of 45 pixels and at -160 and 20 degrees at a distance of 24 pixels. The mortar primitives form a rectangular grid pattern using the relationships at 90, -90, and 0 degrees. Note that the distance of 132 pixel at 0 degrees is reasonable because the image is only 256x256 pixels. Hence 2 complete mortar primitives at approximately 145 pixels across could not both fit along the same horizontal line within the image.

The disparity of primitive sizes, 40 pixels long for primitive 2 and 145 pixels long for primitive 1, will lead to many different angular relationships between primitives of type 1 and 2. It is necessary only to verify that they are related in some way so that the 2 grids can be interposed at the proper angle and distance. The strongest relationship is at -30 degrees and a distance of 12 pixels. At this angle and distance it would seem that the 2 primitive types actually form alternate rows. Using the primitive analysis information in figure 2b the textural pattern can be filled in to produce figure 5b.

The systematic interpretation of the relational results of figures 5a and 6a has not yet been automated and the patterns displayed in figures 5b and 6b have been generated by hand. For such a scheme to be successful the patterns under analysis must be homogeneous regular texture patterns.

Conclusion

Work on this program is still in progress. It is hoped that the relational information produced will lend itself to automatic interpretation. This would lead to the generation of more definitive descriptions of homogeneous regular textures.

References

[1]. R. Nevatia, K. Price and F. Vilnrotter, "Describing Natural Textures," USCIPI Semiannual Technical Report #860 march 1979, pp. 29-54.

[2]. F. Vilnrotter, R. Nevatia and K. Price, "Automatic Generation of Natural Texture Descriptions," USCIPI Semiannual Technical Report #910, September 1979, pp. 31-63.

[3]. F. Vilnrotter, R. Nevatia, and K. Price, "Extraction of Texture Primitives," USCIPI Semiannual Technical Report #960, March 1980, pp. 48-59.

[4]. R. Nevatia, K. Price, and F. Vilnrotter, "Describing Natural Textures," Proc. of the Sixth IJCAI-79, Tokyo, Japan, August 1979.

[5]. F. Vilnrotter, R. Nevatia, and K. Price, "Structural Description of Natural Textures," To appear in Proc. of the Fifth IJCPR-80, Miami, Florida, December 1980.

2.3 Non-Supervised Learning by SVD for Texture Analysis*

B. Ashjari

Introduction

Singular value decomposition is a technique of matrix transformation which has been used as a linear algebraic method for obtaining least square solutions for a set of homogeneous equations [1]. SVD has also been used for calculating the pseudo-inverse of a matrix [2]. In the field of image processing where large matrices of data such as pictures are dealt with, SVD can play an important role in extracting the most useful information from an exorbitant amount of highly correlated data. SVD, therefore, provides a tool for image compression and efficient picture transmission [3].

*This section describes previously unreported results of earlier work supported by DARPA.

In pattern recognition, SVD has been explored for automatic image feature extraction and texture classification [4-5]. In this report, a powerful technique for non-supervised learning and classification by SVD will be discussed.

The mathematical definition of SVD is

$$\underline{F} = \underline{U}\ \underline{S}\ \underline{V}^T \tag{1}$$

where, the matrix $\underline{F}$ is decomposed into three matrices $\underline{U}$, $\underline{S}$, and $\underline{V}$. $\underline{U}$ and $\underline{V}$ are unitary (for complex $\underline{F}$), or orthonormal (for real $\underline{F}$) and $\underline{S}$ is a real diagonal matrix which contains the positive singular values of $\underline{F}$ in descending order. Singular values are excellent descriptors of elemental inter-relationships in a matrix or picture which is non-structural. An example of a structural image is that of man-made objects; and an example of a non-structural one includes textural images such as pictorial patterns of grass, raffia, sand, or wool.

One of the best models for image texture is a stochastic one. In considering a stochastic model for the texture field $\underline{F}$, the elements of $\underline{F}$ can be considered as random variables with correlation among all of them. Using equation (1), for decomposing $\underline{F}$, the elements of $\underline{U}$, $\underline{S}$ and $\underline{V}$ matrices will also be random variables. It has been mathematically proved and experimentally verified that variations of elements in the texture field $\underline{F}$ is substantially galvanized around its singular values rather than affecting elements of $\underline{U}$ and $\underline{V}$ [5]. This latter property is used here for non-supervised learning.

Two sets of experiments, one with simulated texture and the other with natural textures, have been performed which will be elaborated on.

Method of Evaluation

For evaluating the performance of SVD learning, Bhattacharyya distance figure of merit is used. The procedure is as follows:

i) The textures' histograms are gaussianized to have zero mean and unit variance. By taking this measure, all photographic biases are removed and the textures will only differ in their internal shape.

ii) Non-overlapping 32x32 sample windows are extracted from each textural image.

iii) For every sample window, a vector of singular values is computed.

iv) Each vector of singular values is normalized.

v) From each normalized vector of singular values, mean, deviation, skewness and kurtosis is obtained and a 4-dimensional feature vector is formed [6].

vi) For each texture field, there are N feature vectors where N is the number of sample windows. Using these feature vectors, the sample mean and sample covariance are computed.

vii) Using the sample mean and covariance of each pair of texture fields, their B-distance is computed.

Optimum Number of Samples

An experiment to determine the optimum number of samples has been performed with 32, 64, 128 and 196 samples. It has been found that 64 gives the optimum number of samples.

Computer Simulation of Texture

The objective of this section is to generate 512x512 texture fields which are zero mean, normally distributed, having a separable first order Markovian covariance matrix with parameter ρ, where ρ can be given a desired value between 0 and 1. For visualizing purposes, a linear transformation is performed on each generated field to make it a 8-bit (integer) picture with dynamic range of 0 to 255. Figure 1 shows 6 simulated textures with normal distribution and correlation in row and column directions such that

$$K_C = K_R = \begin{bmatrix} 1 & \rho & \rho^2 & \cdots & \rho^{31} \\ \rho & 1 & \rho & \cdots & \rho^{30} \\ \rho^2 & \rho & 1 & \cdots & \rho^{29} \\ \vdots & \vdots & \vdots & & \vdots \\ \rho^{31} & \rho^{30} & \rho^{29} & \cdots & 1 \end{bmatrix} \tag{2}$$

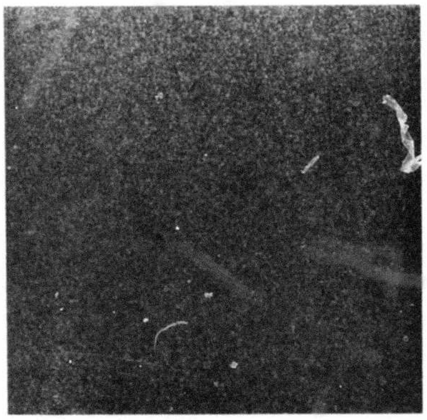
K_C and K_R are column and row covariance matrices and are of toeplitz form which implies fist order Markov behavior [7].

Unsupervised Learning with Simulated (Artificial) Texture

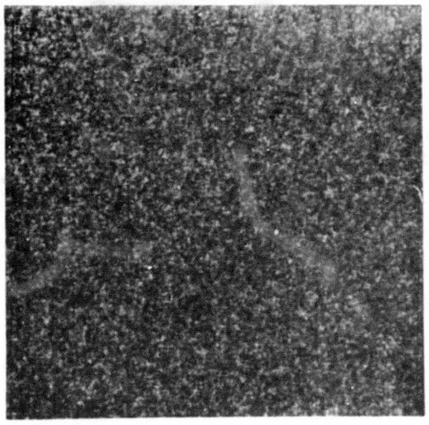
After generating the texture fields with various ρ's, 16 32x32 sample windows of texture with $\rho=0.5$ is imbedded onto texture fields with $\rho=0.6$, $\rho=0.7$, and $\rho=0.9$. The same embedments are performed in a reverse manner.

NOTATIONAL CONVENTION 1:

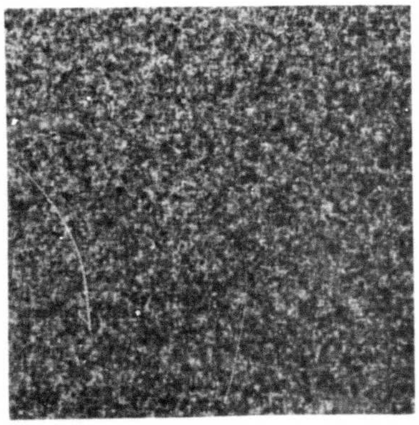
"0.5" of figure 1(b) represents the texture associated with $\rho=0.5$.
"0.5/0.9" of figure 2(e) means the texture associated with $\rho=0.9$ has been imbedded in the texture associated with $\rho=0.9$.



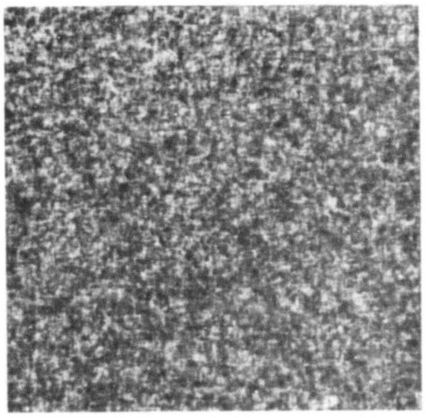
(a) "0.0", ρ=0.0



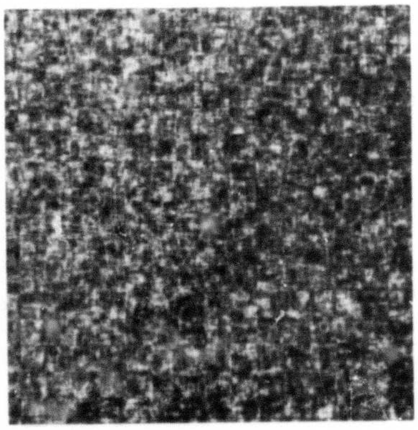
(b) "0.5", ρ=0.5



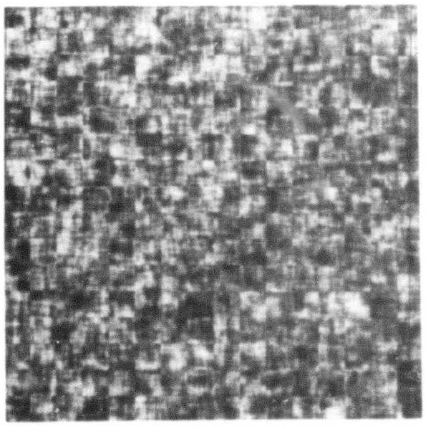
(c) "0.6", ρ=0.6



(d) "0.7", ρ=0.7



(e) "0.8", ρ=0.8



(f) "0.9", ρ=0.9

Figure 1. Computer Simulation of Texture.

Figure 2 presents the imbedded textures for which B-distances between various combinatorial pairs have been tabulated in Tables 1, 2, and 3.

REMARK 1: From each of the texture fields in Figure 2, 64 32x32 sample windows are randomly selected making sure that the 16 imbedded ones are included! This, in essence, means that for example, "0.5/0.9" provides 75% "0.5" + 25% "0.9" for B-distance computation.

REMARK 2: Each 32x32 texture field has mean zero. Their second moment is $tr(K_C \otimes K_R) = tr(K_C) x tr(K_R)$. For a toeplitz matrix of 32x32, the trace is equal to 32. Hence, the second moment becomes

$$\text{Second Moment} = 32 \times 32 = 1024$$

We have 256 of these matrices resulting in the second moment of the whole 512x512 texture to be

$$\text{Total Second Moment} \approx 256 \times 1024 = 262144$$

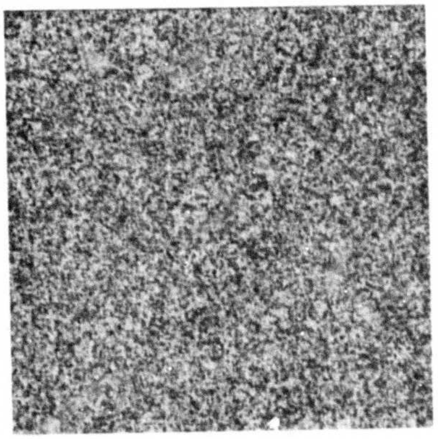
Therefore, we can conclude that all texture fields have the same mean (0.0) and variance (262144.0).

Unsupervised Learning with Natural Texture

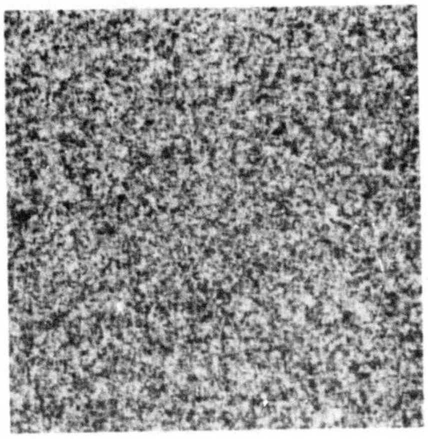
It has been found out that among pairs of grass, raffia, sand, and wool textures, the highest B-distance belongs to the raffia-wool pair [6]. In this set of experiments, 16 sample windows of wool are imbedded onto the raffia image and vice versa. Then, the B-distances are computed as in the simulated case. Figure 3 displays raffia, and wool and their imbedded combinations.

NOTATIONAL CONVENTION 2:

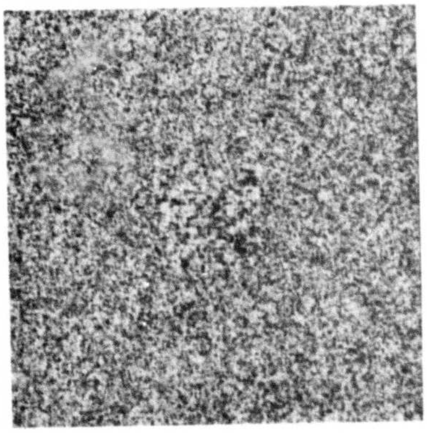
Raffia/wool means 16 samples of wool are imbedded onto raffia



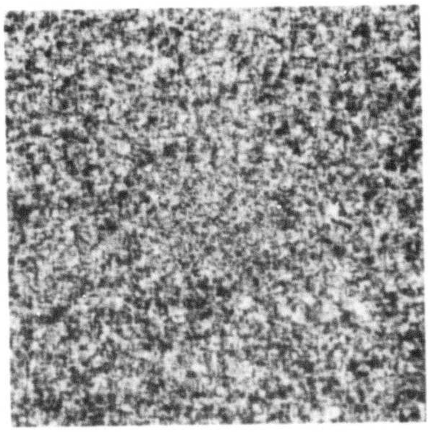
(a) "0.5/0.6"



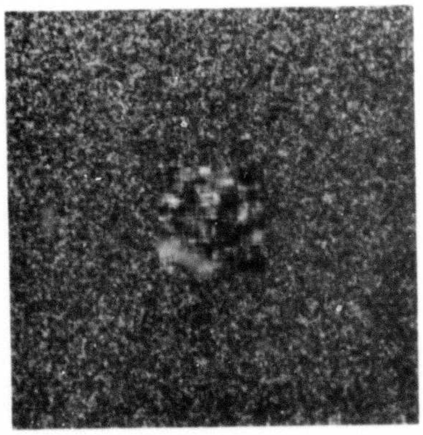
(b) "0.6/0.5"



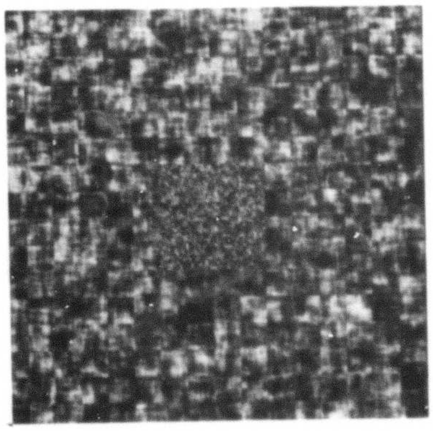
(c) "0.5/0.7"



(d) "0.7/0.5"

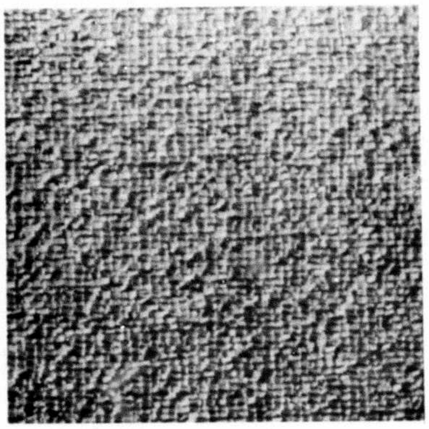


(e) "0.5/0.9"

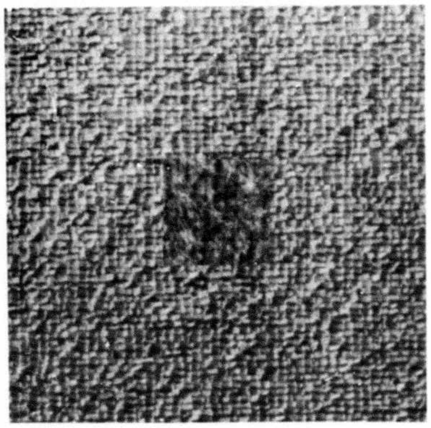


(f) "0.9/0.5"

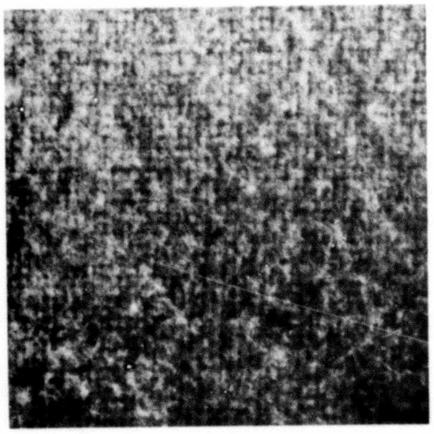
Figure 2. Imbedded Simulated Textures.



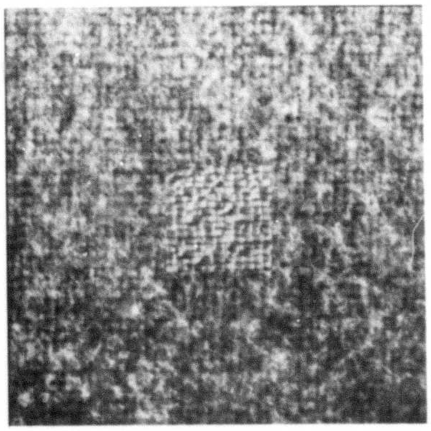
(a) Raffia



(b) Raffia/wool



(c) Wool



(d) Wool/raffia

Figure 3. Natural textures and their imbedded version.

PAIR OF TEXTURES		B - DISTANCE 64 SAMPLES
"0.5"	"0.5/0.6"	0.1716
"0.5"	"0.6"	1.5860
"0.5"	"0.6/0.5"	0.5559
"0.5/0.6"	"0.6"	0.4813
"0.5/0.6"	"0.6/0.5"	0.1338
"0.6"	"0.6/0.5"	0.1183

Table 1. B-distances for various combinatorial pairs of simulated textures, "0.5", "0.5/0.6", "0.6", and "0.6/0.5".

PAIR OF TEXTURES		B - DISTANCE 64 SAMPLES
"0.5"	"0.5/0.7"	0.5376
"0.5"	"0.7"	6.8913
"0.5"	"0.7/0.5"	1.1731
"0.5/0.7"	"0.7"	0.9745
"0.5/0.7"	"0.7/0.5"	0.2080
"0.7"	"0.7/0.5"	0.3605

Table 2. B-distances for various combinatorial pairs of simulated textures, "0.5", "0.5/0.7", "0.7", and "0.7/0.5".

background. Upon extracting sample windows, this embedment is equal to 75% raffia + 25% wool.

Table 4 presents B-distances for various possible pairs.

REMARK: Each texture field is real picture with zero mean, unit variance, and gaussian histogram.

Analysis

The analysis is presented for Table 3 for artificial textures and Table 4 for natural ones. Table 3 is chosen because it provides higher B-distances; and "0.5/0.9" and "0.9/0.5" are easy to visualize. However, the same analysis can be given for Tables 1 and 2.

The B-distance between "0.5" and "0.5" is obviously zero. Table 3 shows that B-distance between "0.5" and "0.9" is 47.1449 which is considered to be enormous. The same table displays B-distance between "0.5" and "0.5/0.9" (i.e. 75% "0.5" + 25% "0.9") is 1.5284. B-distance between "0.5" and "0.9/0.5" (i.e. 25% "0.5" + 75% "0.9") is 2.5166. The drop from 47.1449 to 2.5166 is clearly an indication that a great change in B-distance happens when 75% of "0.9" is used instead of 100%. The drop is 47.1449-2.5166=44.6283 and is so great that the bound of error in telling it is very close to zero. Likewise, when "0.5" is taken with 100% of "0.5" the B-distance is zero; but when "0.5" is taken with 75% "0.5" + 25% "0.9", B-distance increases to 1.5284. The classification error bound for the latter is 8%. This means that the two textures which used to be the same can now be separated with a good probability of 92% due to existence of a foreign texture in one of them.

This is an indication of the usefulness of singular value decomposition for non-supervised learning and classification. Table 4 displays that B-distance between raffia and 100% wool is 11.8746. The distance drops sharply to 1.6211 when 75% wool + 25% raffia is

PAIR OF TEXTURES		B - DISTANCE
		64 SAMPLES
"0.5"	"0.5/0.9"	1.5284
"0.5"	"0.9"	47.1449
"0.5"	"0.9/0.5"	2.5166
"0.5/0.9"	"0.9"	1.7812
"0.5'0.9"	"0.9/0.5"	0.3439
"0.9"	"0.9/0.5"	0.9081

Table 3. B-distances for various pairs of simulated textures "0.5", "0.5/0.9", "0.9", and "0.9/0.5".

PAIR OF TEXTURES		B - DISTANCE
		64 SAMPLES
Raffia	Raffia/Wool	0.7128
Raffia	Wool	11.8746
Raffia	Wool/Raffia	1.6211
Raffia/Wool	Wool	1.8350
Raffia/Wool	Wool/Raffia	0.5033
Wool	Wool/Raffia	0.9625

Table 4. B-distances for various pairs of simulated textures, Raffia, Raffia/Wool, Wool and Wool/Raffia.

employed instead of 100% wool. The foreign texture (i.e. raffia) introduced in wool causes that drop in the B-distance. This is also clearly an indication that singular value decomposition can be used as a means for non-supervised learning for natural textures.

As it can be observed from figure 2(a) and (b), the embedments of "0.5/0.6" and "0.6/0.5" are barely noticeable by human eye. However, Table 1 displays that, through the application of the technique presented in this report, these embedments can automatically be distinguished by computer.

References

[1]. G.H. Golup and C. Reinsch, "Singular Value Decomposition and Least Square Solutions," Numerish Matematic, Vol. 14, 1970, pp. 403-420.

[2]. G.H. Golup and W. Kahan, "Calculating the Singular Values of Pseudo-inverse of a Matrix," Journal of SIAM Numer. Anal., Ser. B, Vol. 2, No. 2, 1965, p. 250.

[3]. H.C. Andrews and C.L. Patterson, "Outer Product Expansion and their uses in Digital Image Processing," IEEE Trans. on Computers, Vol. C-25, No. 2, February 1976, pp. 140-148.

[4]. B. Ashjari and W.K. Pratt, "Singular Value Decomposition Image Feature Extraction," USCIPI Report 800, Image Processing Institute, USC, Los Angeles, Calif., March 31, 1978, pp. 72-89.

[5]. B. Ashjari, Singular Value Decomposition Image Feature Extraction, Ph.D. Dissertation, Image Processing Institute, USC, Los Angeles, Calif., to be published.

[6]. B. Ashjari and W.K. Pratt, "Supervised Classification with Singular Value Decomposition Texture Measurement," USCIPI Report 860,

Image Processing Institute, USC, Los Angeles, Calif., March 31, 1979, pp. 52-62.

[7]. W.K. Pratt, Digital Image Processing, Wiley-Interscience, New York, 1978.

2.4 A Best-Fit Model Approach to Markov Texture Synthesis

D.D. Garber and A.A. Sawchuk

Introduction

A method of generating texture simulations according to their Nth order densities was investigated for binary textures in an earlier paper [1]. The simulations resulting from this markov process resembled quite closely their parental textures in most cases. When applying a similar concept to multi-grey level imagery, the limits of computer storage are soon reached. To circumvent this constraint, a new method of texture synthesis was developed and applied to a small number of textures. The results to present are given in this paper.

N-grams in Continuous Imagery

In the binary texture generation based on N-grams a single value $P(V_{N+1}/V_1,...,V_N)$ was stored for each possible pattern $(V_1,V_2,...,V_N)$ where $V_i=0,1$ in the binary case. This value represented the conditional probability that the next pixel, V_{N+1}, in the generation process would be a 0 or black pixel. The V_i's were chosen by a best linear model fit detailed in [2] and therefore the kernel of previous pixels $(V_1...V_N)$ is not necessarily contiguous (see Figure 1). Details concerning the estimation of $P(V_{N+1}/V_1,...,V_N)$ from a parent

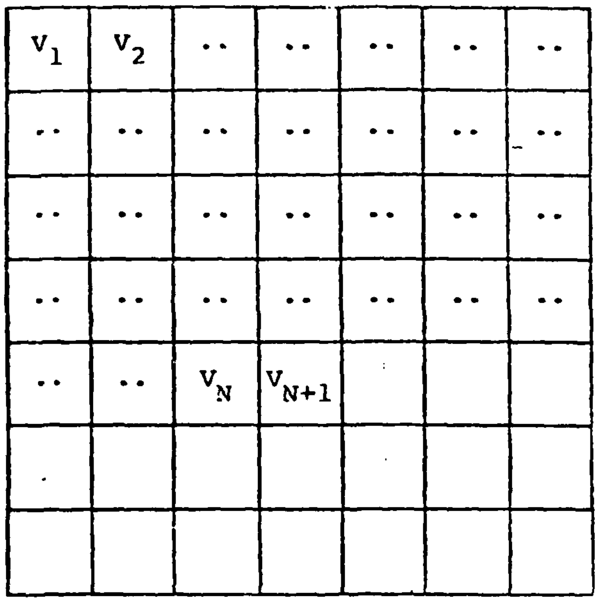


Figure 1a. Two-dimensional synthesis (contiguous kernel).

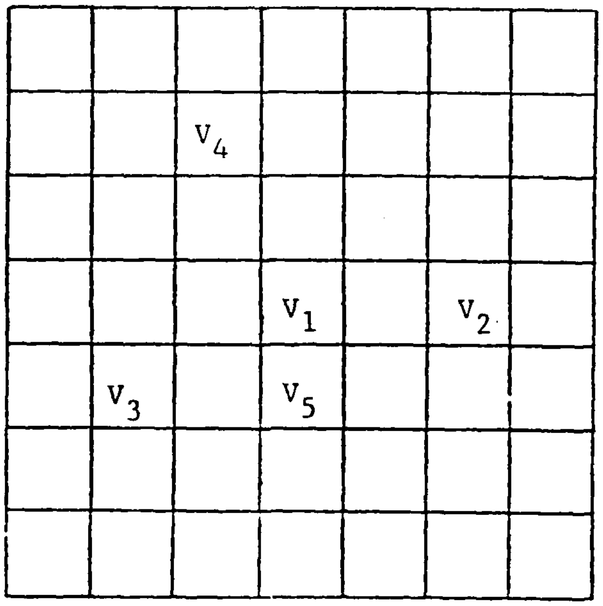


Figure 1b. Two-dimensional synthesis (non-contiguous kernel).

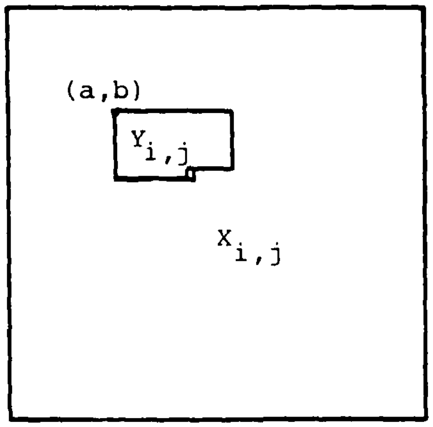


Figure 2. Passing kernel over parent texture.

texture are given in [1]. This single value is sufficient in a binary case to define the distribution of V_{N+1} given $V_1,\ldots,V_N$. The number of these values which we are required to store is 2^N. In the generation process each pixel V_{N+1} is generated based on the values of the pixels $V_1,\ldots,V_N$ surrounding it and on a computer-generated uniformly-distributed random variable. The texture simulations are generated pixel by pixel along a row until each row is complete. Pixel generation along the edges of an image can be handled in a variety of ways but are normally assumed to be any random value, 0 or 1, if they are outside the image boundaries.

A similar approach could be used to generate multi-grey-level textures. For a grey-level texture, G^{N+1} values of $P(V_{N+1}/V_1,\ldots,V_N)$ must be stored. (Actually only $(G-1)\cdot G^N$ are required as $\sum_{X=0}^{G-1} P(X/V_1,\ldots,V_N)=1$ for all V_i). Storage limitations are soon reached. Also estimation of $P(V_{N+1}/V_1,\ldots,V_N)$ is difficult as multiple occurrences of the pixel pattern $V_1,\ldots,V_N$ may not exist in the parent texture. Therefore even without storage limitations the problems of estimating $P(V_{N+1}/V_1,\ldots,V_N)$, which represents the distribution of V_{N+1} given the values of $V_1,\ldots,V_N$ is complex.

This estimation problem no doubt has a number of ad hoc solutions. The problem is basically that for large N and or large G, there may not be a suitable number of occurrences of the pattern $V_1,\ldots,V_N$ for certain values of V_i to adequately estimate the distribution $P(V_{N+1}/V_1,\ldots,V_N)$ given a finite sample size. Even though a certain pattern never occurs or rarely occurs in our sample parent texture it is not implied that such a pattern is impossible and will never occur in our simulation synthesis. We might often find numerous occurrences of this pattern if our sample size or the size of our parent texture was increased, especially in noisy and fine-structured textures. But as this very large sample may not be present, we must estimate $P(V_{N+1}/V_1,\ldots,V_N)$ for all $V_1,\ldots,V_N$ based on available samples.

One approach would be to use sample patterns which closely resemble but which may not be exactly the same as each pattern $(V_1,\ldots,V_N)$. That is in a pictorial sense, we use patterns of $(V_1,\ldots,V_N)$ which look "close to" the pattern for which we are attempting to estimate $P(V_{N+1}/V_1,\ldots,V_N)$. Therefore samples in our sample parent texture may be used to estimate numerous $P(V_{N+1}/V_1,\ldots,V_N)$ and not just those they fit exactly. But the concept of "close to" must be first numerically defined. Given two patterns, one from our sample texture and the other from the conditional probability we are attempting to estimate, a distance measure used to define "close to" could be used to determine the value of that sample in estimating $P(V_{N+1}/V_1,\ldots,V_N)$. If the fit between the kernel pattern and the pattern of interest is good the associated value of V_{N+1} in the parent texture will be valuable in estimating $P(V_{N+1}/V_1,\ldots,V_N)$.

Normally, when N and G are small or we have a large number of samples and numerous samples for any given $V_1,\ldots,V_N$ occur, we would merely use the histogram of the associated V_{N+1} to estimate $P(V_{N+1}/V_1,\ldots,V_N)$. Here the relative number of times a particular value of V_{N+1} occurs given a pattern indicates the conditional probability we are attempting to estimate. Where a distance measure is used instead, a good fit could be considered to be synonymous with high frequency of that pattern and a poor fit with low frequency.

If such a method of estimating these conditional probabilities is used we are still faced with a huge storage problem. To be practical, the storage requirement must be reduced. From an information standpoint, it is interesting to note that a method of estimating N-grams or conditional probabilities $P(V_{N+1}/V_1,\ldots,V_N)$ from a sample parent texture image produces G^{N+1} data values from M^2 pixel samples where M is the size of the square parent texture image in pixels. For large G and N this is a drastic increase in data. But the actual information content can really never be greater than that content of the sample parent texture image. Therefore, this M^2 value represents an upper bound on the amount of data we should use to generate a texture simulation. Any amount of data exceeding this will contain redundant, useless data.

The Synthesis Method

Combining this concept of upper bound with the idea of forming a distance measure to compare two texture kernel values leads to a new texture synthesis method. In this method, we can generate the next pixel based on the pixels in the kernel surrounding it (see Figure 1) and their comparison to similar kernels in the parent texture. This comparison can be accomplished by passing the kernel currently present in the simulation process, over the parent texture and computing the distance function at all possible points (see Figure 2). Denoting the pixels in the parent texture by X_{ij}, $i,j=0,\ldots,M-1$ and the pixels in the kernel $V_1,\ldots,V_N$ by $Y_{i,j}$, we can compute a comparison image

$$C_{a,b} = \text{COMPARISON}(X_{i+a,j+b}, Y_{i,j}) \tag{1}$$

for all a,b such that the kernel is within the boundaries of the texture.

One possible comparison function would be correlation. Assuming, without loss of generality, that our kernel is contiguous as in Figure 3 and the elements are denoted as Y_{ij}, this function would be defined as

$$r_{a,b} = \frac{\sum_i \sum_j X_{a+i,b+j} Y_{i,j} - \frac{\left[\sum_i \sum_j X_{a+i,b+j}\right]\left[\sum_i \sum_j Y_{i,j}\right]}{N}}{\left[\left[\sum_i \sum_j X_{a+i,b+j}^2 - \frac{(\sum_i \sum_j X_{a+i,b+j})^2}{N}\right]\left[\sum_i \sum_j Y_{i,j}^2 - \frac{(\sum_i \sum_j Y_{i,j})^2}{N}\right]\right]^{\frac{1}{2}}} \tag{2}$$

The problem with this particular distance measure is quite serious. Correlation does not take into account differences in over-all mean. For example, the kernels in Figure 3 are perfectly correlated but their means differ significantly.

A better choice would be the mean square difference (MSD). This may be defined as

$$MSD_{a,b} = \sum_i \sum_j (X_{i+a,j+b} - Y_{i,j})^2 \qquad (3)$$

where i,j must be within the coordinate range of the kernel as in Eq. (1). This particular comparison function weights the comparison of all elements in a kernel equally. Having studied many texture generation models we immediately recognize that this fit is not properly weighted. The few pixels which are closest to Y_{next} in proximity are far more important when predicting Y_{next} than those which are far away. So Eq. (2) may be modified to give the weighted mean-square difference (WMSD)

$$WMSD_{a,b} = \sum_i \sum_j (X_{i+a,j+b} - Y_{i,j})^2 \cdot W_{i,j} \qquad (4)$$

A choice of W would be

$$W_{i,j} = \frac{1}{(i-i_{next})^2 + (j-j_{next})^2} = \frac{1}{R^2} \qquad (5)$$

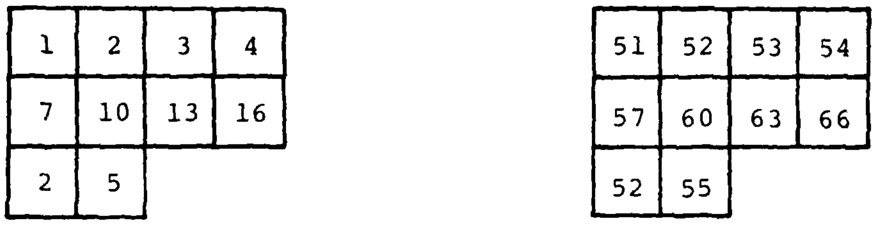


Figure 3. Perfectly correlated kernels.

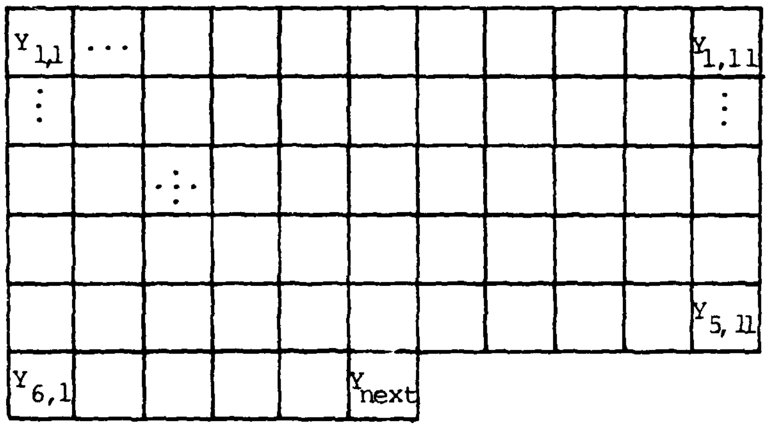


Figure 4. Best-fit model kernel (contiguous).

where R is the euclidean distance between pixel $Y_{i,j}$ and the kernel eye Y_{next} and the coordinates of the eye are given by (i_{next}, j_{next}).

As the first step in comparing a given kernel $Y_{i,j}$ to all kernels in the parent texture, for each point (a,b) in the parent texture, ignoring edges, the WMSD is computed resulting in an image of WMSD's. Where the fit between the generated kernel $Y_{i,j}$ and the image $X_{i,j}$ is good, we would expect $WMSD_{a,b}$ to be small. The smallest WMSD represents the "best" fit according to our norm. We could choose the Y_{next} associated with this best fit at point (a,b) to be our next pixel in the generation process, however this can cause problems. First of all, the generation process would "lock in" on the parent texture and the generated texture could very well become just an exact copy of the input parent texture. Second, we know ideally that Y_{next} has a distribution, not just a mean. In the autoregressive model [3] we gave Y_{next} a distribution by adding random noise to it. We could do that here. However we are failing to use additional information in the WMSD image. There maybe a set of points (a,b), all exhibiting a good fit to the kernel pattern $Y_{i,j}$. In fact, the best fit may have a noisy Y_{next} and the other good fits could provide information to improve the prediction of the Y_{next} in the generation process. Using a set of best fits is equivalent to increasing our sample size. We look at a set of similar patterns to pick our Y_{next}.

At this point there are numerous ways to proceed. Logically those patterns with the "best" fit should provide better estimators for Y_{next} so some kind of weighting decision is needed to choose the relative importance of the WMSD's found. If we search through the WMSD image and find the minimum value, $WMSD_{min}$, and scale all the WMSD's by that we form a new image MAX1

$$MAX1_{a,b} = \frac{WMSD_{min}}{WMSD_{a,b}} \tag{6}$$

This image has the value 1.0 at the best fit point and values $0 \le MAX1 \le 1.0$ elsewhere.

Here we can look at the MAX1(a,b) image and study its range. If $0.16 \le MAX1 \le 1.0$ it is implied that the worst fit gives a 0.16 MAX1(a,b). Somehow that worst fit should be translated to imply that the probability of choosing the Y_{next} associated with that point $(a,b)_{worst}$ is nearly 0.0. The simplest way of doing that is to take powers of the image MAX1(a,b). The maximum remains 1.0 while smaller numbers approach 0.0. For example $(1.0)^{10}=1.0$ but $(0.16)^{10}=1x10^{-8}$. We do this to obtain some kind of estimate of $P(Y_{next}/Y_{i,j})$. After studying the values of MAX1(a,b) and its powers, the value of 16 was chosen and a new image PDFUNS

$$PDFUNS_{a,b} = (MAX1_{a,b})^{16} \tag{7}$$

was chosen to estimate the probability density function $P(Y_{next}/Y_{i,j})$. PDFUNS can be scaled so that $\sum_a \sum_b PDFUNS(a,b)=1$. Then a uniformly distributed random variable, r, [0,1] is generated and a point (c,d) is found such that

$$\begin{aligned} \sum_{a=1}^{c-1} \sum_{b} PDFUNS_{a,b} + \sum_{b=1}^{d-1} PDFUNS_{c,b} &< r \\ \sum_{a=1}^{c-1} \sum_{b} PDFUNS_{a,b} + \sum_{b=1}^{d} PDFUNS_{c,b} &> r \end{aligned} \tag{8}$$

The Y_{next} associated with the kernel shape at (c,d) is then used as the next pixel in the generated image. The process is continued until a full texture image is generated with the kernel window moving one pixel at each step, row by row.

Results

For a kernel containing 55 pixels (see Figure 4) passed over a 128x128 parent texture approximately 7.2×10^{6} operations (additions or subtractions) are needed to get the $WMSD_{a,b}$ image defined by Eq. (4). Another 2.6×10^{5} are required to find the next pixel according to Eq. (8). therefore, to generate a 512x512 texture requires only 1.96×10^{12} (2 trillion) operations.

Results from texture synthesis done by this method are shown in Figure 5 through 9. Each of these images is 512x512 pixels. A 128x128 section of each original (parent) texture was used for the simulation. Bark exhibits very large macro structure and this is lost in the simulation. A similar thing happens with raffia as the kernel size is smaller than the cell size of the original texture but is not as pronounced. The top part of the bubbles texture was generated using a 128x128 portion different than that of the bottom part. For this reason the top 20-30% of the texture looks different from the rest. The large number of operation makes this process very time consuming even where a pipelined processor is dedicated to the task. About 5.5 days of dedicated time on an AP120B are required to generate each texture.

This method is of little use at present due to the computational complexity of the algorithm but a few points should be brought out. With constantly increasing computer processing speeds, a simplified version of this texture simulation method may be implemented in the near future. It is even possible that such computations could be performed by a properly constructed architecture of micro processors. In any case such brute-force approaches are in many aspects simple and could be made cost-effective in the future. The results also indicate

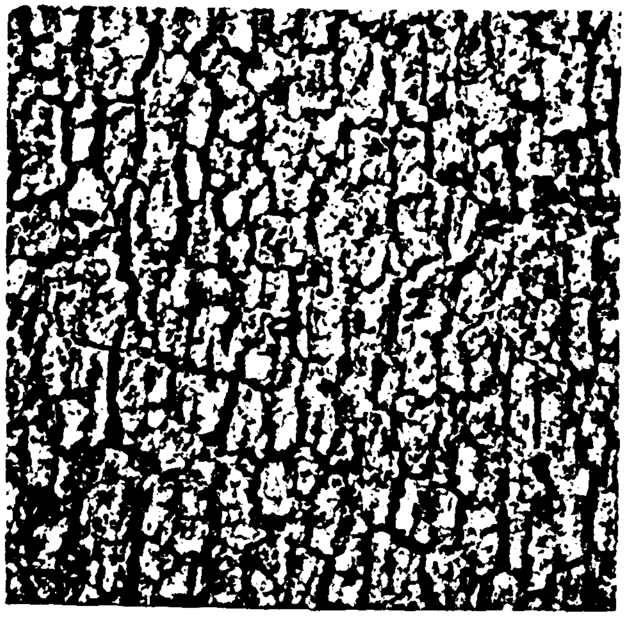


Figure 5a. Original bark.

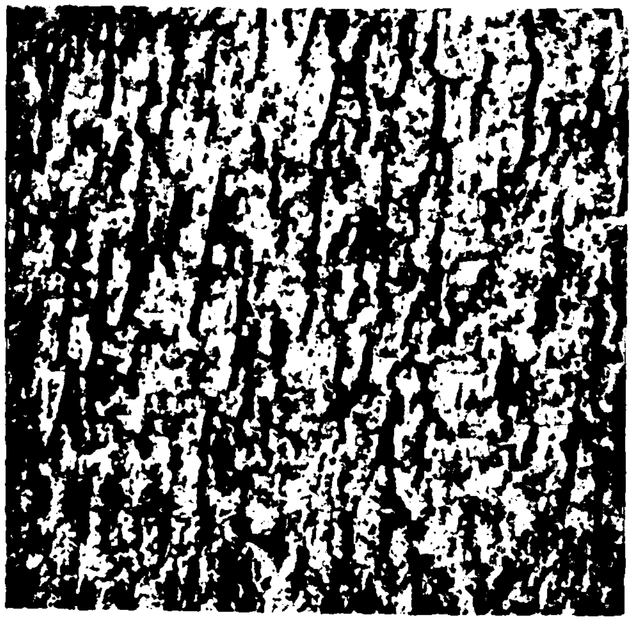


Figure 5b. Simulated bark.

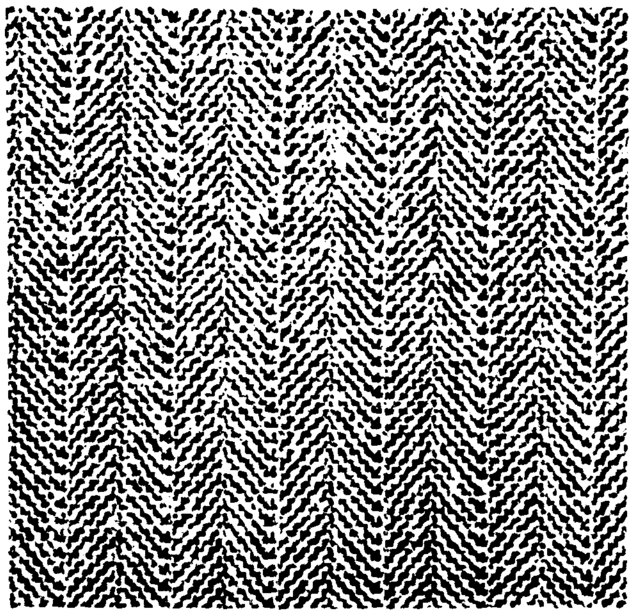


Figure 6a. Original cloth.

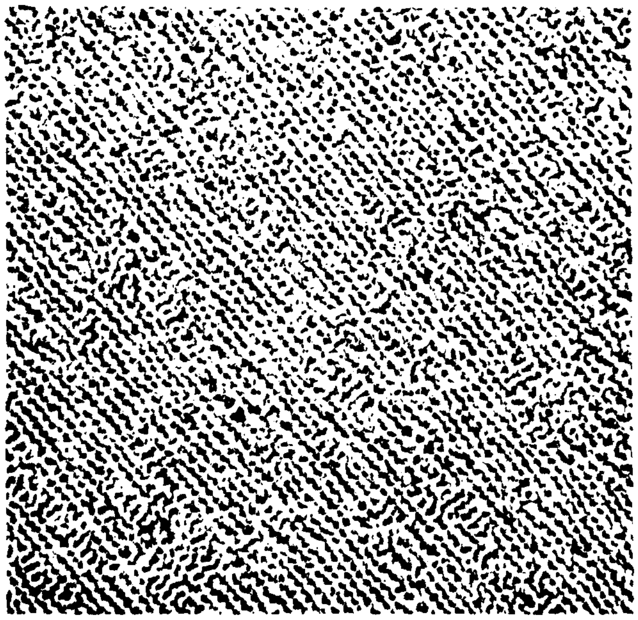


Figure 6b. Simulated cloth.

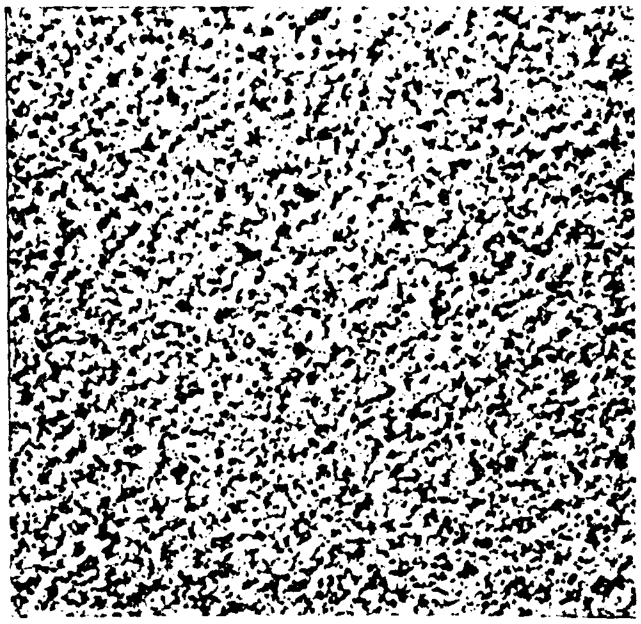


Figure 7a. Original sand.

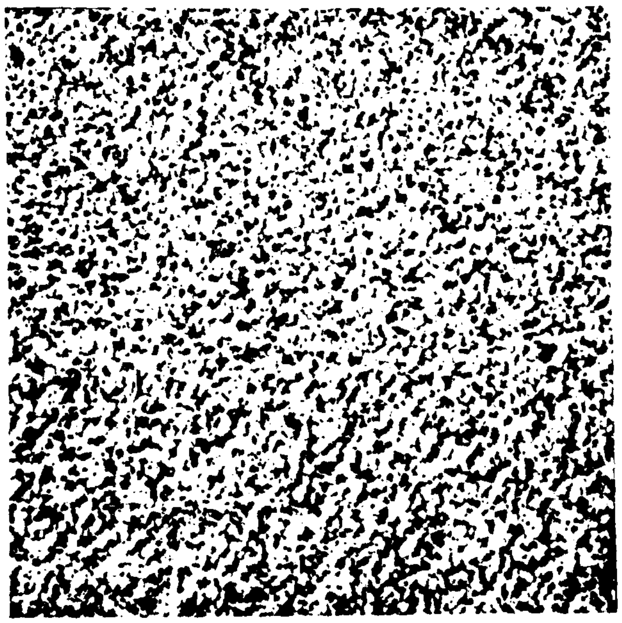


Figure 7b. Simulated sand.

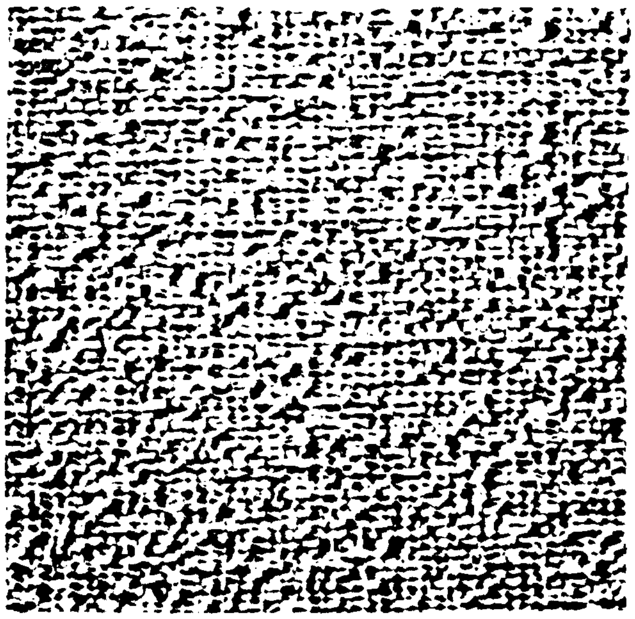


Figure 8a. Original raffia.

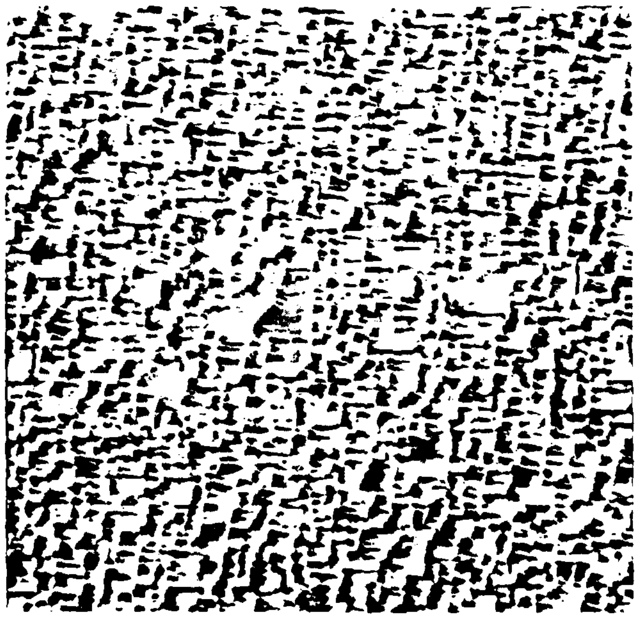


Figure 8b. Simulated raffia.

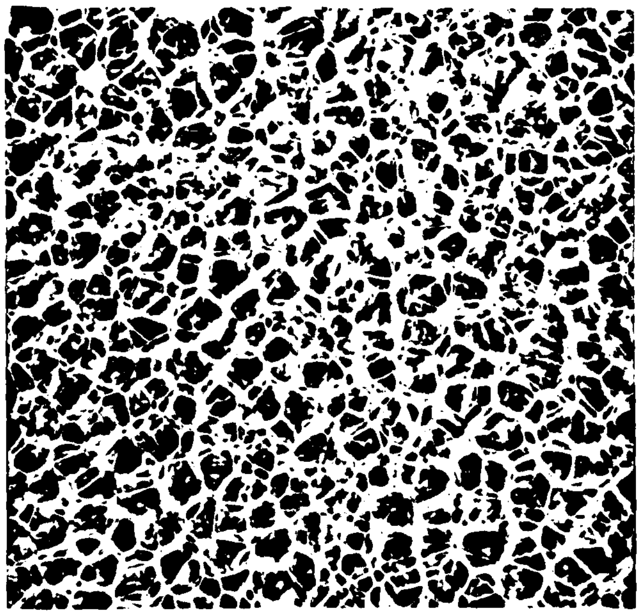


Figure 9a. Original bubbles.

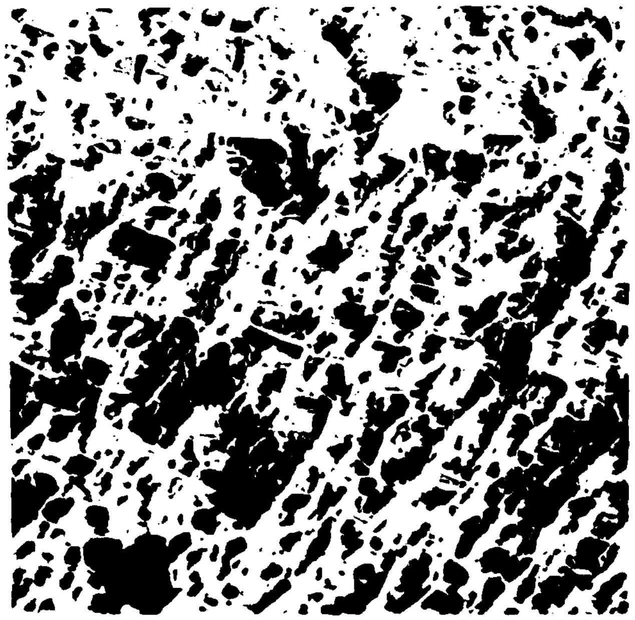


Figure 9b. Simulated bubbles.

visually the amount of texture information present in a 55 pixel window (see Figure 4) as at each pixel generation step, the next pixel in the Markov process depends on only the pixels in this neighborhood.

Finally, this approach is admittedly ad hoc. Numerous distance measures could replace the one chosen in this work and each would give different, either better or worse, results. It is always important that the process be random and not merely copy the texture sample. This type of non-random process will generate patterns quickly observable if the simulation is much larger than the original. In other words, the histogram represented by $P(V_{N+1}/V_1,...,V_N)$ should rarely be a delta function. A reduction in the number of computations could be made if the kernel was non-contiguous. Also, better results could probably also be obtained if the kernel window was larger. The shape, contiguity and size of the kernel in this study was chosen primarily for computer programming considerations.

Conclusion

The results from this best-fit texture synthesis method are very pleasing but the number of computations required is large. Other similar algorithms could be developed which are simpler and could possibly produce even better results. With the decrease in computation costs and the increase in processor speeds of future computers, such texture synthesis methods could be easily implemented in the future.

References

[1]. D.D. Garber, "Models for Texture Analysis and Synthesis," USCIPI Report 910, 1979.

[2]. D.D. Garber, "Application of the General Linear Model to Binary Texture Synthesis," USCIPI Report 960, 1980.

[3]. D.D. Garber and A.A. Sawchuk, "Higher-Order Texture Synthesis Models and Residual Examination," USCIPI Report 960, 1980.

2.5 Additional Texture Synthesis Models and Results

D.D. Garber and A.A. Sawchuk

Introduction

In earlier work, a second-order linear texture synthesis model was proposed. In this paper additional results of simulations using that model are given. Also, some results of simulations using non-gaussian, non-normal noise are presented. Synthesis methods designed to generate textures regardless of magnification are developed. Preliminary results on use of multiple linear models for texture synthesis and the use of the covariance matrices to segment textured images are briefly discussed. A method for making textures non-stationary through post-processing is also examined.

Second-Order Markov

First-order linear autoregressive models were among the first approaches to texture synthesis [1,2]. The results were appealing and it was believed that further improvements could be made by expanding to a second-order model [3]. Initial results showed only a slight visible improvement on the texture sand (see Figure 2). It was felt that additional work should be done on other textures to determine the improvement due to the second-order model and that texture models employing non-gaussian and possibly non-stationary noise be investigated based on the results in [3].

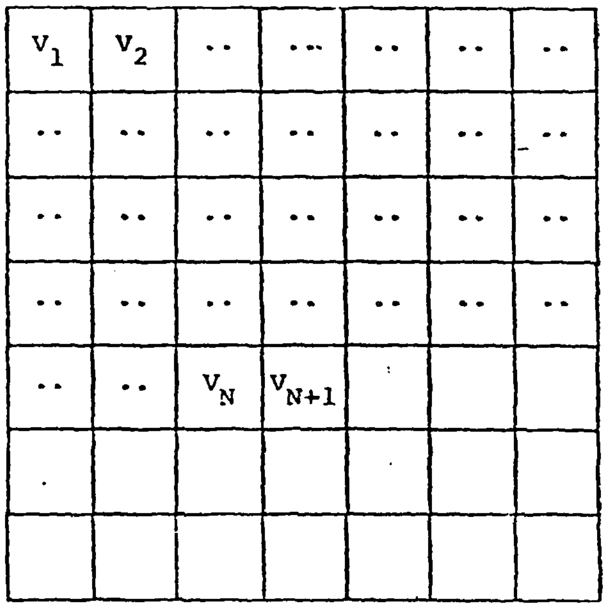
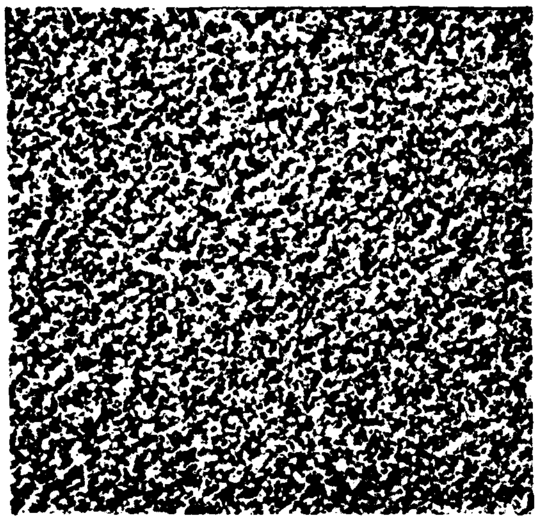
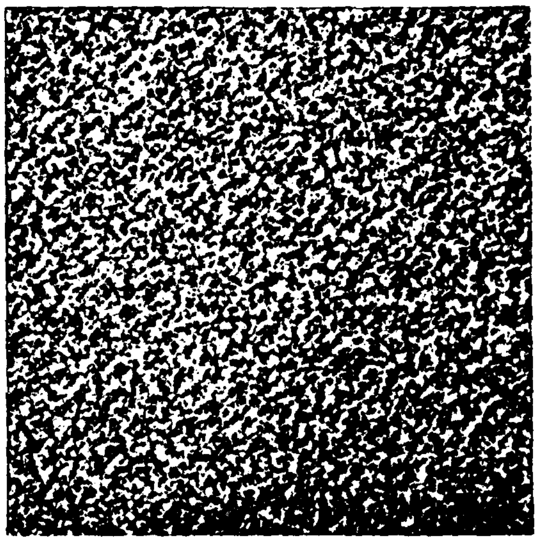


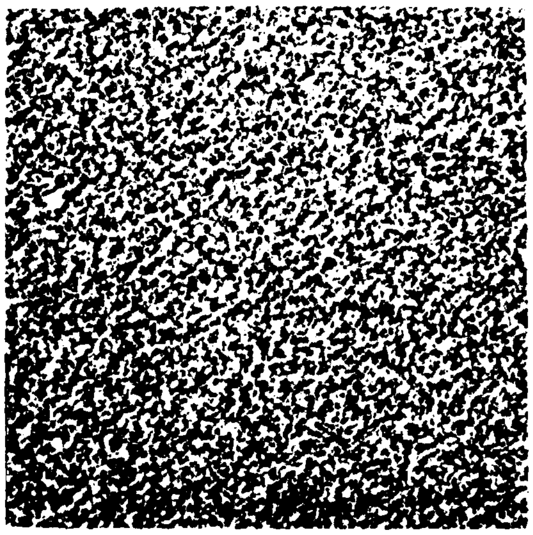
Figure 1. Two-dimensional synthesis (contiguous kernel).



(a) Original SAND.



(b) Simulated SAND with cross-products.



(c) Simulated SAND with non-stationary noise.

Figure 2.

In the linear autoregressive model, each pixel, V_{N+1}, in the synthesized texture is produced by computing a linear combination of the pixels V_i in the kernel surrounding it (see Figure 1) plus noise ε. That is

$$V_{N+1} = \beta_1 V_1 + \beta_2 V_2 + \ldots + \beta_N V_N + \beta_0 + \varepsilon. \tag{1}$$

The noise ε was assumed to be independent, stationary gaussian noise with zero mean and fixed variance. A linear second-order model is formed by adding all possible cross-product terms and the model becomes

$$\begin{aligned} V_{N+1} &= \beta_1 V_1 + \beta_2 V_2 + \ldots + \beta_N V_N + \beta_0 + \beta_{11} V_1^2 + \beta_{12} V_1 V_2 + \ldots + \beta_{NN} V_N^2 + \varepsilon \\ &= \sum_{i=1}^{N} \beta_i V_i + \sum_{i=1}^{N} \sum_{j=1}^{N} \beta_{ij} V_i V_j + \beta_0 + \varepsilon \end{aligned} \tag{2}$$

Adding second-order terms to a model will always produce a fit as good as or better than a first-order model but the number of computations required to compute the coefficients and fit the model are much greater.

Inside a circular radius of 14 pixels from V_{N+1} there are 307 pixels. To search all possible cross products in this region to find the most significant would require over 47,000 cross products to be examined. Computation of a covariance matrix containing all of these terms is impossible (in practice). In our study we were limited to investigate only 820 possible cross products for entry into the generation model. As most of the variance was explained by the linear terms of the model, most of the cross products were insignificant from a statistical point of view. This selection procedure is detailed in [2]. Those that were significant were entered into the model and a new texture was generated using Eq. (2) with stationary gaussian noise and with zero mean and fixed variance. The results are shown in

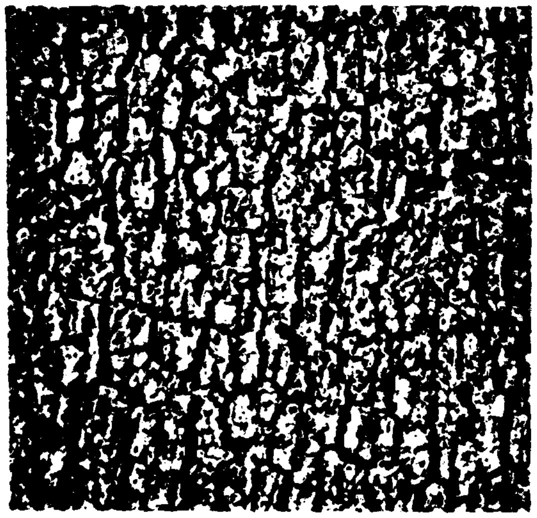
Figures 2 through 5.

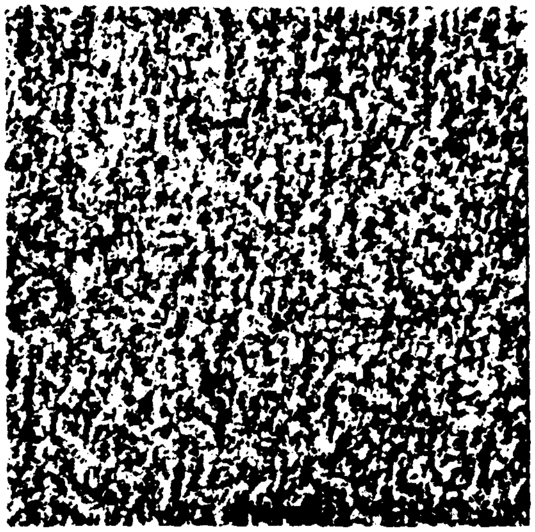
Applying this model to the original image data gives a residual error image. The distribution of this error and the relationships between the predicted and actual pixel values can be studied. Based on this work a method of generating textures using non-stationary noise was developed. A histogram of error as a function of predicted value, $\hat{V}_{N+1}$, can be formed. This, in turn, can be used to generate errors, ε, during the synthesis process. That is, at each pixel a V_{N+1} is predicted according to Eq. (2) excluding the error term and associated with that predicted value is a distribution of error (which is likely to be non-normal and even non-zero in mean) from which a random error value can be chosen to be added to V_{N+1}. The next pixel will then be computed in a similar manner. Results of texture synthesis formed using this model are shown in Figures 2-5.

In most cases, considerable improvement is seen when these simulations are critically observed. Of course, the information required to generate them is considerably greater also. The distribution of errors as a function of $\hat{V}_{N+1}$ must be condensed and coded to some degree to minimize storage requirements. For a 256-grey-level image $\hat{V}_{N+1}$ can range from -50 to 305 and the errors from -255 to +255. This would yield quite a large amount of data if fully stored. By storing a small number (under 100), typical errors for each range (and not each single value) of $\hat{V}_{N+1}$ the number of data values we are required to store can possibly be reduced to under 1000. More experimentation on these coding techniques remains to be done.

Skip-Generate Method

Simulating textures which have a fine structure is a much easier process than simulating textures with coarse structure. This is because the linear model will contain fewer terms if the texture pixels become uncorrelated over a small distance. For the same texture at a greater magnification, the pixels become highly


(a) Original BARK.


(b) Simulated BARK with cross-products.

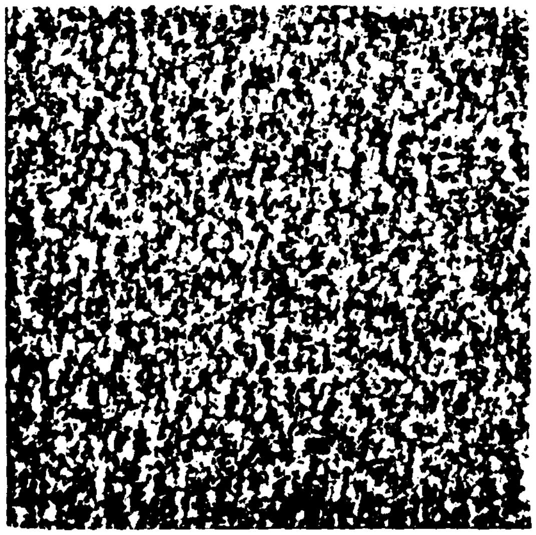
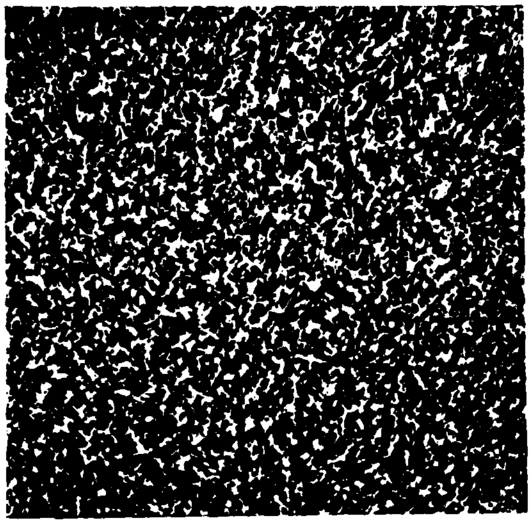
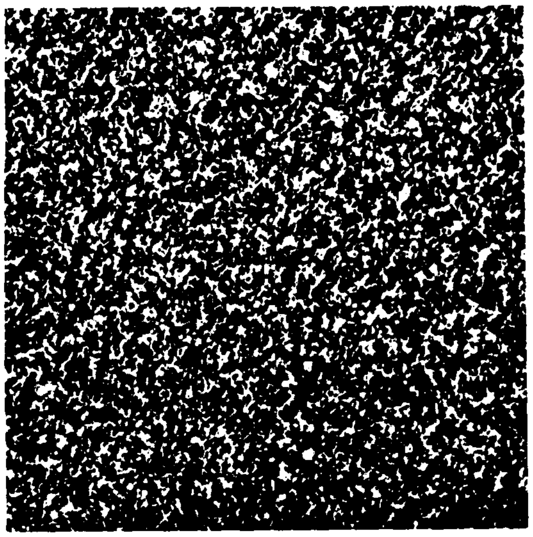

(c) Simulated BARK with non-stationary noise.

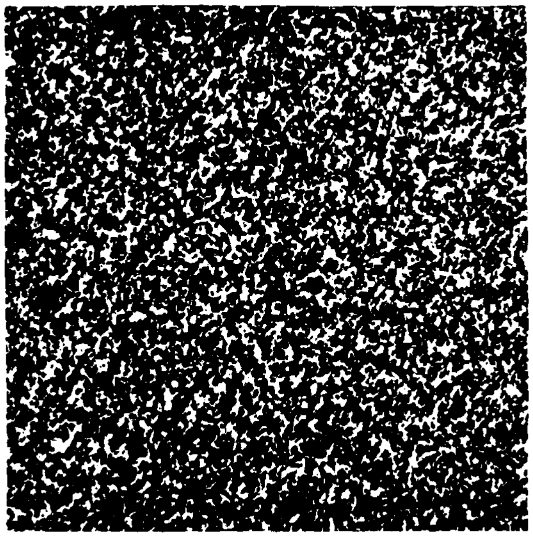
Figure 3.



(a) Original GRASS.

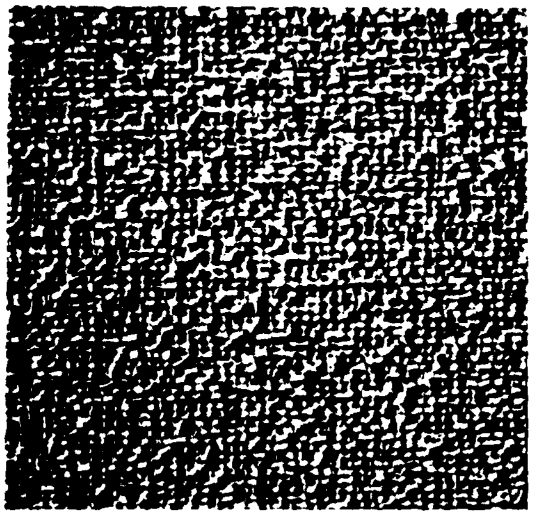


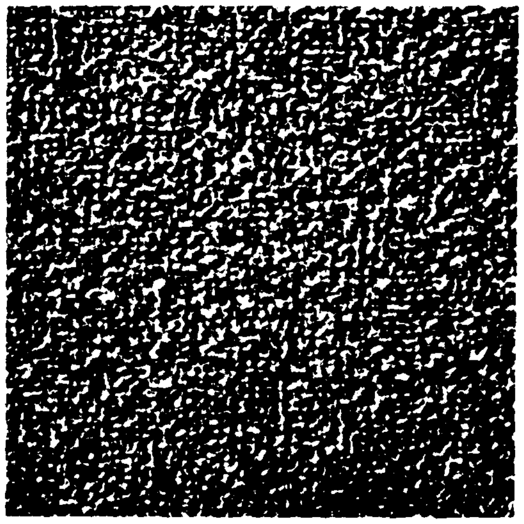
(b) Simulated GRASS with cross-products.



(c) Simulated GRASS with non-stationary noise.

Figure 4.


(a) Original RAFFIA.


(b) Simulated RAFFIA with cross-products.

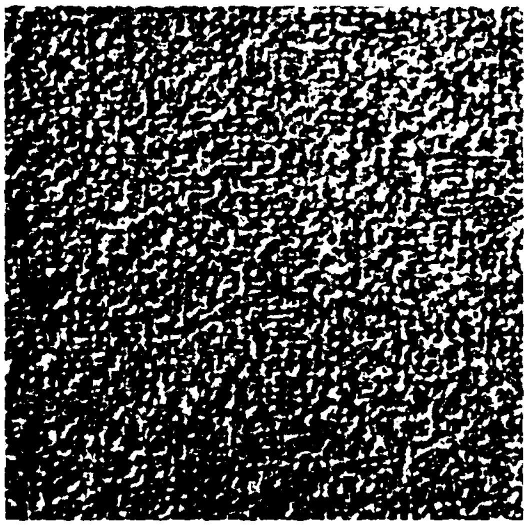

(c) Simulated RAFFIA with non-stationary noise.

Figure 5.

correlated and the linear model will be forced to contain more terms. As the texture becomes more coarse, more time-consuming statistical measurements must be taken on the parent texture over larger windows. To circumvent this problem, a new texture generation method must be developed.

In the texture work so far [1-3], pixel V_{N+1} was generated based on pixels above or to the left of it (see Figure 1). As was discussed in [2], the kernel does not have to be contiguous. This shape is chosen to insure that the image space of our synthesized texture was filled during the generation process. However, generating pixels along a row, row by row is not the only way of filling an image space.

Consider the non-contiguous kernel mask in Figure 6. If the spacing between the pixels in this mask is 8, using the linear model in Eq. (1) to generate the right-most pixel in the bottom row, we can generate every 8th pixel along every 8th row. At each step the next pixel is generated based on the previously-generated pixels around it (ignoring boundary conditions). After generating an image with this type of spacing, the pixels midway between the previously-generated pixels on each row may be generated using the mask in Figure 7. In this mask, the pixel with the "x" in it denotes the next pixel, V_{N+1}, to be generated according to Eq. (1). Naturally, the linear model used in this step will have different coefficients than the previous one. It is also interesting to note that new pixels do not depend only on previously generated pixels above them (as with the mask in Figure 1) but depend also on the pixels below them. But still, ignoring boundary conditions, each pixel depends only on previously generated pixels. At the next step a mask similar to that in Figure 8 can be used to fill in the pixels midway between the previously-generated pixels in each column. Again pixels are allowed to depend on pixels around them.

By repeatedly using the masks in Figure 7 and Figure 8 with successively closer and closer pixel spacing, the texture simulation

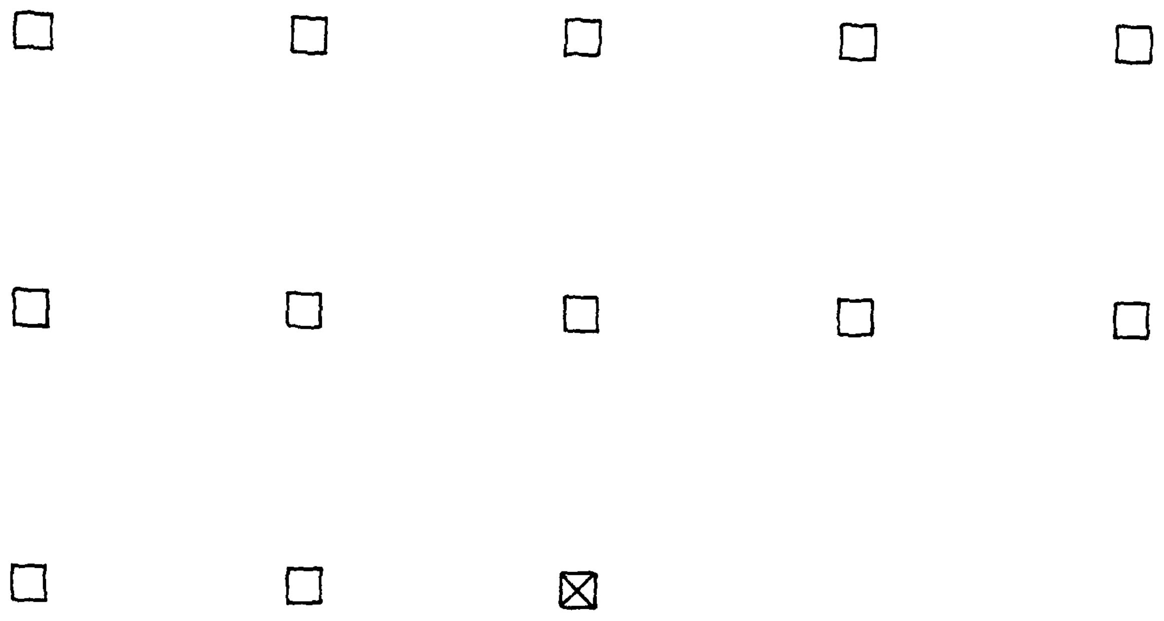


Figure 6. First-Pass Mask.

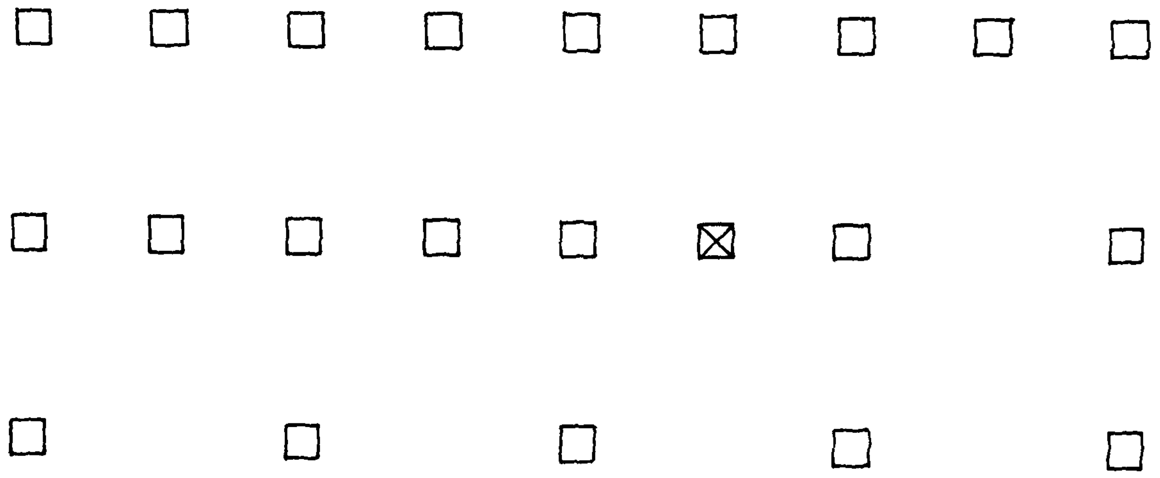


Figure 7. Second-Pass Mask.

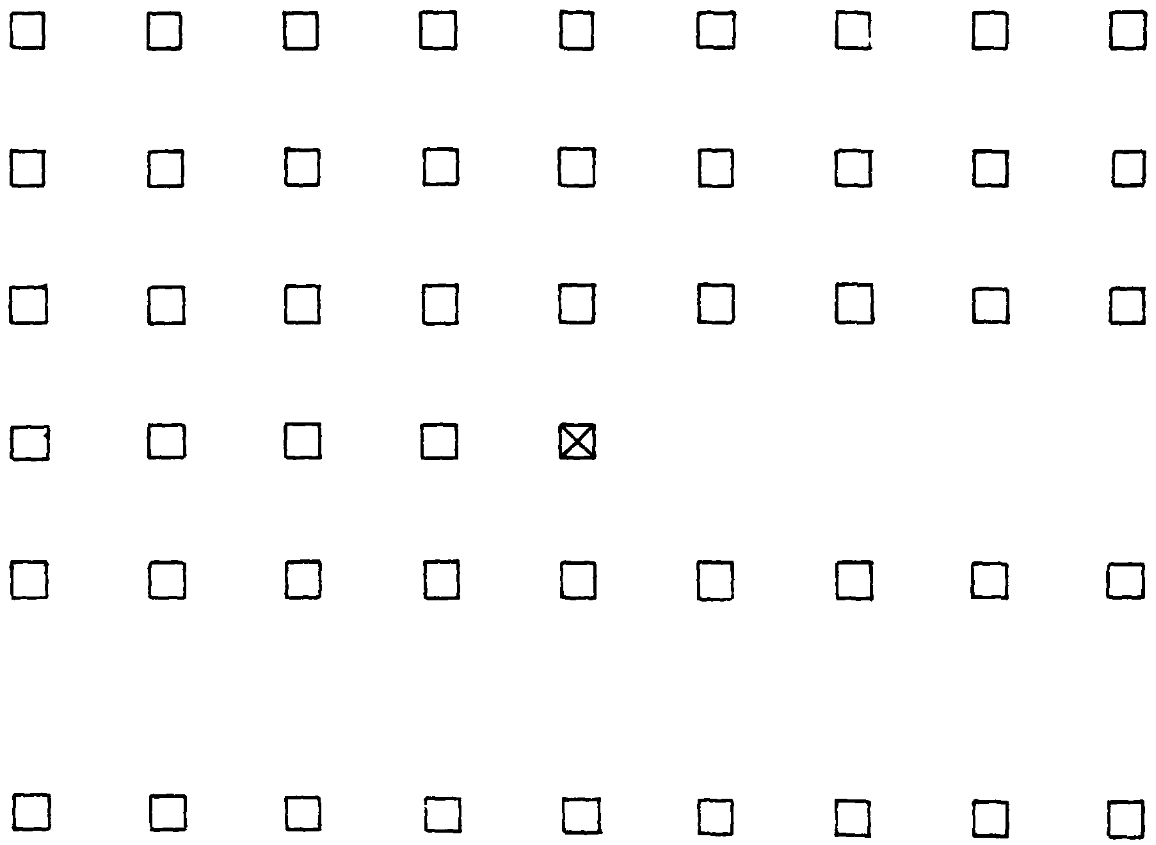


Figure 8. Third-Pass Mask.

```
1 6 4 6 2 6 4 6 1 6 4 6 2 6 4 6 1
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
3 6 4 6 3 6 4 6 3 6 4 6 3 6 4 6 3
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
1 6 4 6 2 6 4 6 1 6 4 6 2 6 4 6 1
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
3 6 4 6 3 6 4 6 3 6 4 6 3 6 4 6 3
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
1 6 4 6 2 6 4 6 1 6 4 6 2 6 4 6 1
```

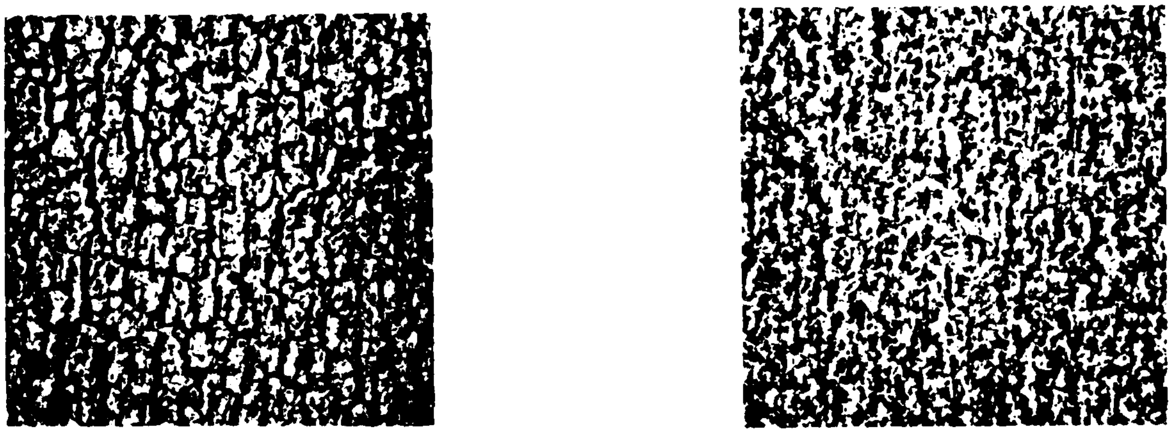
Figure 9. Filled space of skip-generate method.

image space is filled. An example showing the pixels generated at each successive pass is shown in Figure 9. More importantly, to determine the linear model for each mask, only one covariance matrix is required and can contain as many or as few terms as desired. The process of collecting statistics for one matrix is not beyond the complexity that we would want to undertake for the small number of times required by this process. Naturally, any other markov process may be substituted for the one in this linear model. As before, only the measurements required to estimate the parameters corresponding to each mask need to be taken. This number depends on the spacing of the pixels in the first mask, which should be a power of two. Other odd-shaped kernels and kernels whose spacing is not a power of two could be designed to form space-filling sets. Most would require more models to be estimated and would provide little additional information.

Texture simulations using this method are shown in Figures 10-15. Only a slight improvement is seen in some of the texture simulations over the synthesis done by the earlier single linear model. Most of these textures are apparently well simulated by a carefully chosen model and the results are not critically dependent on the coarseness of the textures. More study remains to be done with textures of various magnifications to determine the types of textures where this skip-generate method provides improvement.

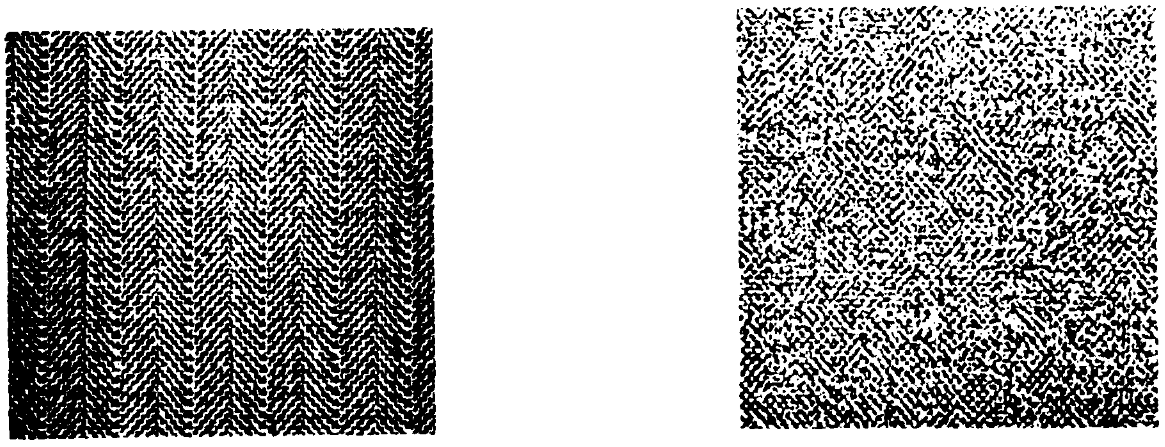
Multiple Linear Model Method

When generating textures using the general linear model described by Eq. (1) and the generating kernel in Figure 1 the same model is used regardless of the values of the pixels $V_1,\dots,V_N$. By developing more than one linear model and allowing the choice of the model at each pixel generation step in the synthesis process to be dependent on some functional value of $V_1,\dots,V_N$, $F(V_1,\dots,V_N)$ a new synthesis model is formed. To illustrate this concept consider the data in Figure 16a. If we were to fit one linear model to the data it would look



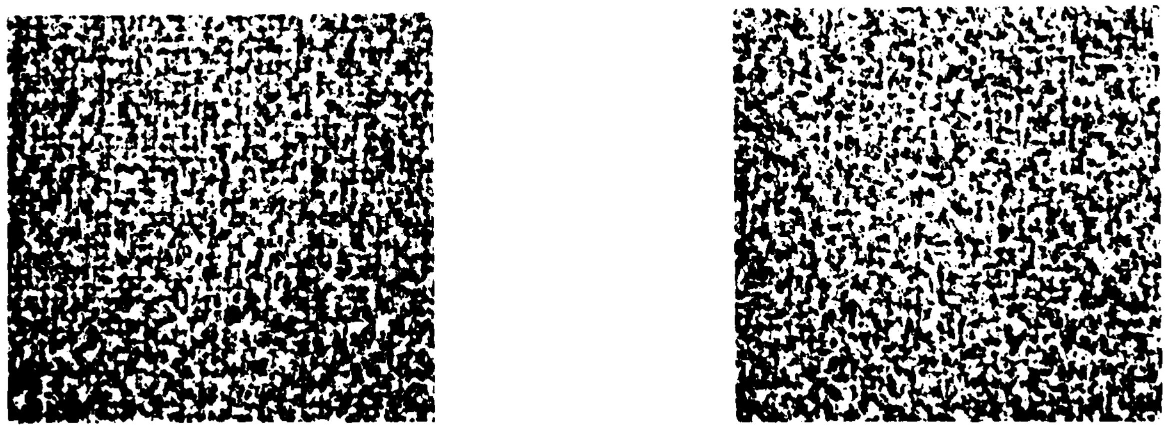
(a) (b)

Figure 10. Bark.



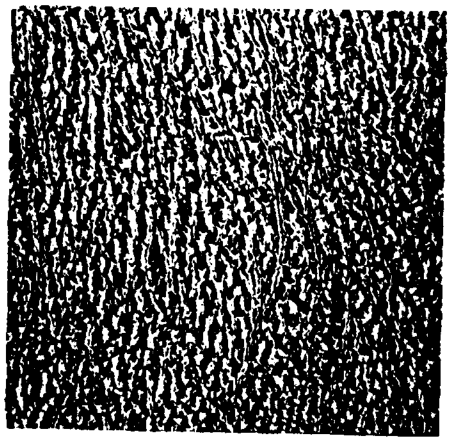
(a) (b)

Figure 11. Cloth.



(a) (b)

Figure 12. Wool.


(a)

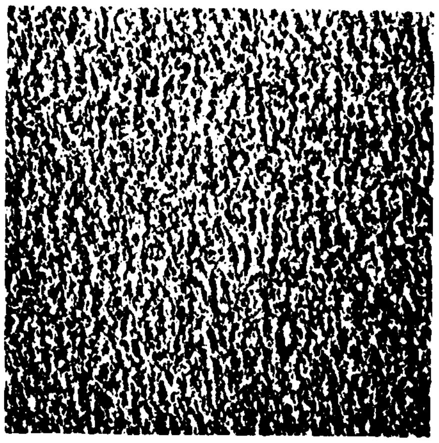
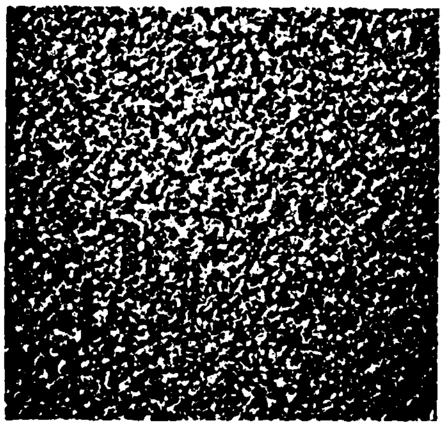

(b)

Figure 13. Leather.


(a)

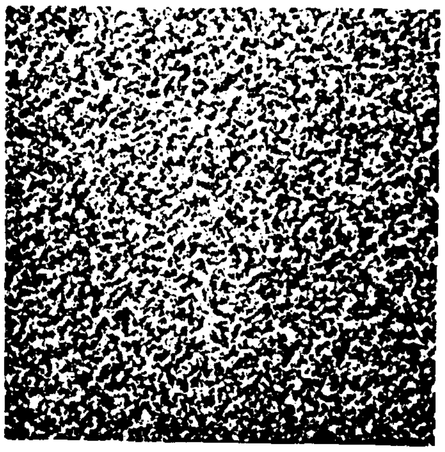
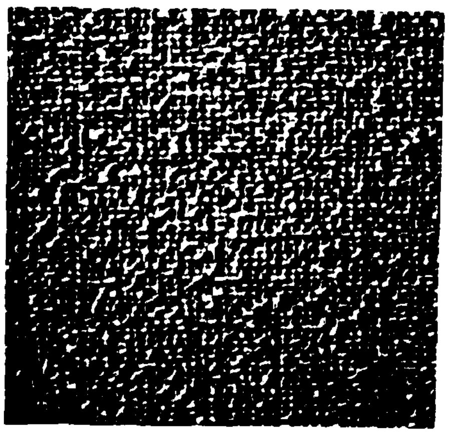

(b)

Figure 14. Sand.


(a)

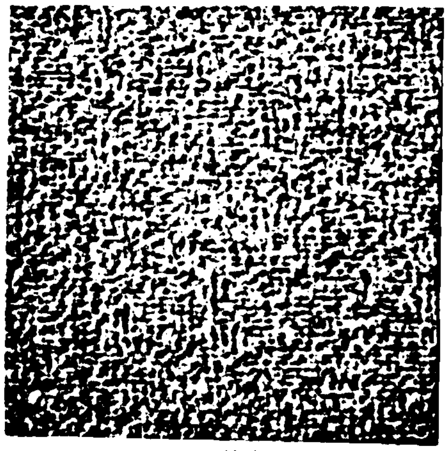

(b)

Figure 15. Raffia.

like the single linear line running through the data. This linear model could then be used to predict V_2 based on the value of V_1 in the typical linear regression way. But if we allow the choice of our linear model to be dependent on the value of V_1 then for an incoming value of V_1 we choose a model whose domain includes V_1. For 6 linear models, the straight lines are shown in Figure 16b. The fit to the data using multiple linear models will always be as good as or better than that of the single linear model. That is, the mean square error will generally be reduced using multiple models.

Using multiple linear models for texture synthesis we would generate pixel V_{N+1} based on pixels $V_1,\ldots,V_N$ in the following way. First, we compute a function, F, of the $V_1,\ldots,V_N$ pixels which allows us to choose the proper linear model. Then using this model with the values $V_1,\ldots,V_N$ we predict V_{N+1} and add noise. This process is diagramed in Figure 17.

Ideally, the function F should be chosen to minimize the total mean square error resulting from fitting the limited number of models to the sample data. This is very difficult to do in practice however as for N larger than 3 we are fitting multiple hyperplanes to data in an N+1 dimensional space.

One texture synthesis of sand was done using the multiple linear model (see Figure 18). In this case eight models were used and the model number was chosen by examining the pixel immediately to the left of the pixel being generated. The range of this pixel, 0 to 255, was divided into 8 equal subranges and the model was chosen according to the subrange into which the value fell. Only a slight improvement over the single linear model synthesis is seen.

Another method of using multiple markov models is to generate an image of fields defining the model number to be used in a second pass. Such an approach would be useful in simulating textures which have multiple sub-textures within them. A simple analytical example may be

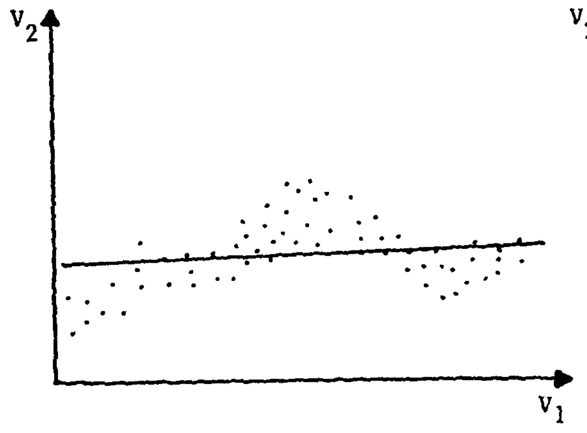


Figure 16a. Single Linear Model.

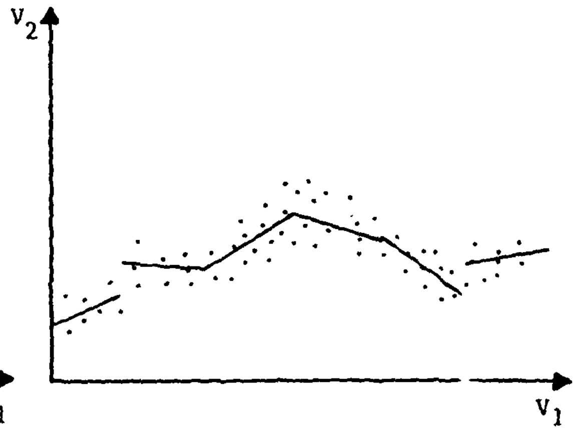


Figure 16b. Multiple Linear Model.

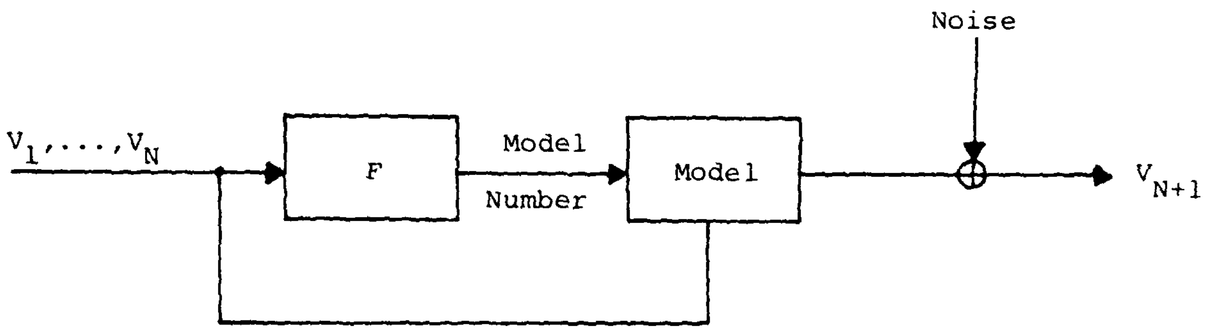


Figure 17. Diagram of Multiple Linear Model.

shown in Figure 19. Real world examples might include such things as a brick wall where the texture of the bricks is different than the texture of the concrete separating them. It was felt that this type of approach might be useful in the simulation of bark which has a strong macro structure. A method to separate this texture into two fields which would later define the model to be used was designed. This result (Figure 20) was obtained by successively passing smart median filters of varying sizes over a binary image (which was obtained by thresholding an original continuous grey-level image) (see Figure 3a). The smart median filters replace the center pixel of a window with the median only if certain conditions are met. In the binary case, the center pixel of a square window is replaced if the percentage of black or white pixels is above a specified threshold. The threshold and window size vary in the successive passes over the image.

In our actual simulation, once this field image is obtained a method must be developed to simulate this field texture and this field texture will then in turn be used to choose the model numbers in the generation of final synthesis. Generating textures with only a few grey levels can be done using more complete markov statistics, perhaps n-grams, but the large size of the fields may require that a method such as the *skip-generate method* be used. This kind of work has not been done yet but is planned in the future.

Generating Non-Homogeneous Textures

Previous to simulation attempts, most textures in our work are preprocessed by statistical differencing (see [4] for details). This processing helps eliminate non-homogeneities in mean which are primarily due to lighting effects. The process of statistical differencing which removes these non-homogeneities can be reversed to induce them back into a synthesized texture.

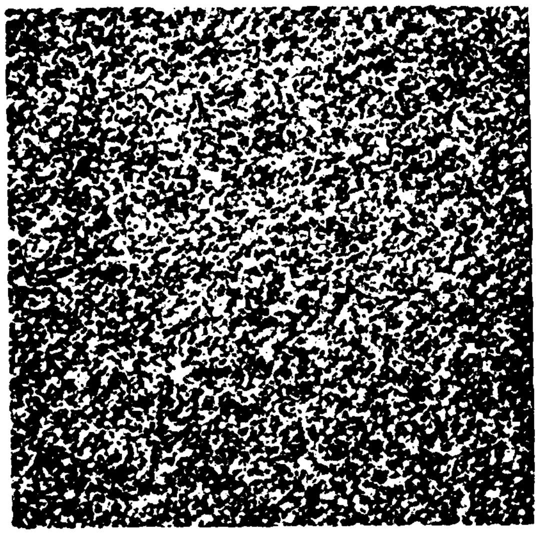


Figure 18. Multiple Linear Model Simulated Sand.

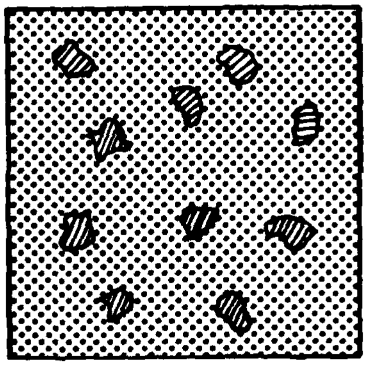


Figure 19. Sub-textures.

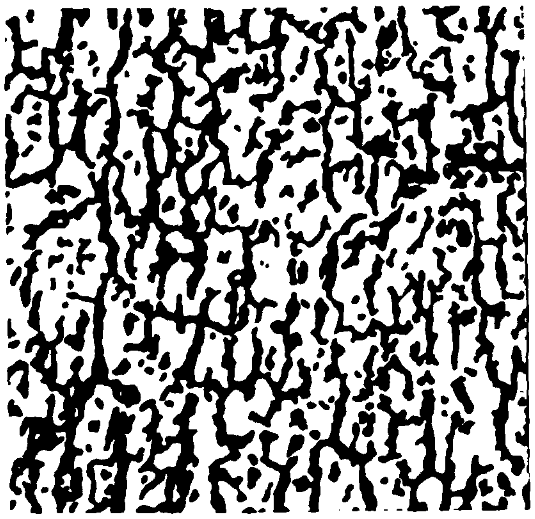


Figure 20. Two Bark Fields.

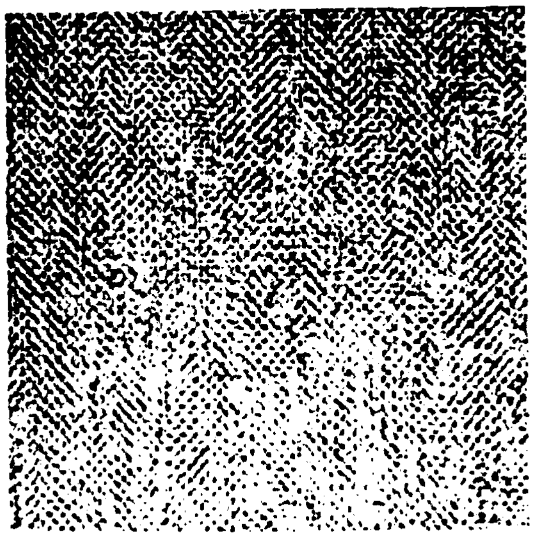


Figure 21. Local Moment Modified Cloth.

Statistical differencing is defined by

$$G(j,k) = [F(j,k)-\overline{F}(j,k)]\left[\frac{A\sigma_d}{A\sigma(j,k)+\sigma_d}\right]+[\alpha M_d+(1-\alpha)\overline{F}(j,k)] \qquad (3)$$

where m_d and σ_d represent desired mean and standard deviation. F is the input pixel at location (j,k) and G is the output pixel in the statistical differenced image. $\overline{F}(j,k)$ and $\sigma(j,k)$ are usually estimated from the input image over a window containing pixel (j,k).

The inverse operation of statistical differencing can be called local moment modification. Solving for F in terms of G we find the formula for local moment modification as

$$F(j,k) = \frac{A\sigma(j,k)+\sigma_d}{A\sigma_d} G(j,k) + \left[\frac{\overline{F}(j,k)A\sigma_d}{A\sigma(j,k)+\sigma_d} - [\alpha M_d+(1-\alpha)\overline{F}(j,k)]\right] \qquad (4)$$

By generating an image $\overline{F}(j,k)$ and $\sigma(j,k)$ and defining A, α, m_d and σ_d, using Eq. (4) we can induce non-homogeneities into our simulated texture.

One such synthesis where $\overline{F}(j,k)$ was assumed to be ramp-like and $\sigma(j,k)$ was a constant is shown in Figure 21. Other more complex and more random $\overline{F}(j,k)$ and $\sigma(j,k)$ images could be used to create different effects.

Segmentation Using Covariance

In [2] details concerning the estimation of a covariance matrix and linear model determination from a sample texture are given. This covariance and the linear model itself may be used to identify and segment textures. A little-known method for testing the equality of t covariance matrices given assumptions of normality is derived using the standard maximum likelihood ratio approach in [5]. The resulting test for t=2 covariance matrices is given by

$$2.3026\ mM \cong \chi^2_{p(p+1)/2} \qquad (5)$$

where

$$M = (n_1+n_2-2)\log_{10}|S|-(n_1-1)\log_{10}|S_1|-(n_2-1)\log_{10}|S_2|,$$

S_1 and S_2 are the covariance matrices for each group,

$$S = [(n_1-1)S_1+(n_2-1)S_2]/(n_1+n_2-2),$$

$$m = 1-[\frac{1}{(n_1-1)}+\frac{1}{(n_2-1)}-\frac{1}{(n_1+n_2-2)}][(2p^2+3p-1)/6(p+1)],$$

and p is the size of the covariance matrices being tested. Here n_1 and n_2 are the sample sizes used to estimate S_1 and S_2. The derivation of Eq. (5) is very complex and will not be detailed in this paper. For fixed sample sizes m is fixed so the primary distance measure is M. This value approaches 0 when the matrices are equal and becomes large the more they differ.

The composite texture shown in Figure 22 was segmented using this approach. A covariance matrix was measured over a 48x48 pixel window and the 16x16 window in the center of it was compared to the known covariance for sand, the center texture in the composite. It should be recalled that sand is not well-described by its covariance matrix as the synthesis of sand using a linear model was poor. The covariance matrix was 16x16 and described the relationships of the pixels in the pattern shown in Figure 24. Statistical measurements taken over a maximum pixel separation of 6 were used to produce a distance measure. A scaled version of the output of the distance measure M is shown in Figure 23. As was expected, the sand region is picked out correctly.

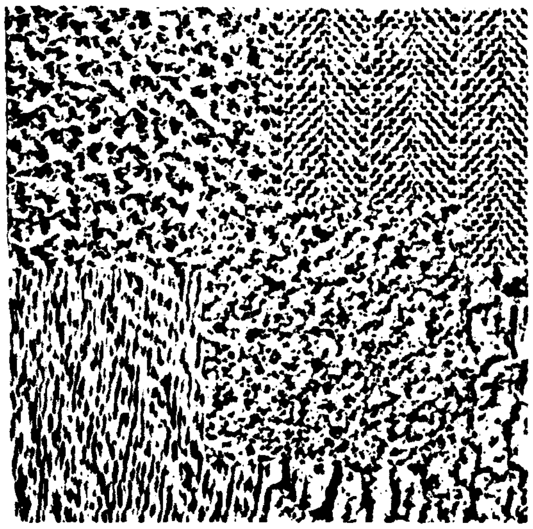


Figure 22. Composite Texture.

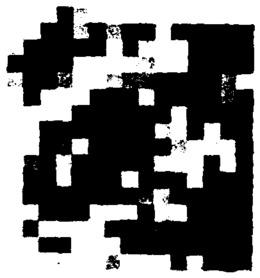


Figure 23. Segmented Composite.

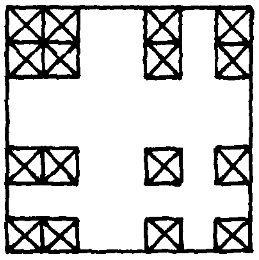


Figure 24. Covariance Matrix Kernel.

There is no doubt that better methods of segmenting and identifying textures exist. The assumptions of normality and independent samples on which this test for equality of matrices depends are clearly violated. This may account for the poor classification results. Better results will be obtained when going to larger matrices perhaps but the computational complexity likewise increases drastically.

Conclusion

New models for texture synthesis were described in this paper along with results. It is felt that many will be useful in the future to do texture simulations. A texture identification and segmentation method was presented with preliminary results.

References

[1]. D.D. Garber, "Models for Texture Analysis and Synthesis," *USCIPI Report 910*, 1979.

[2]. D.D. Garber, "Application of the General Linear Model to Binary Texture Synthesis," *USCIPI Report 960*, 1980.

[3]. D.D. Garber and A.A. Sawchuk, "Higher-Order Texture Synthesis Models and Residual Examination," USCIPI Report 960, 1980.

[4]. W.K. Pratt, "Digital Image Processing," Wiley, 1978.

[5]. M. Kendall and A. Stuart, "Advanced *Theory of Statistics*," Griffin, 1976.

2.6 Runway Detection in Aerial Images of Airports

K.R. Babu

Introduction

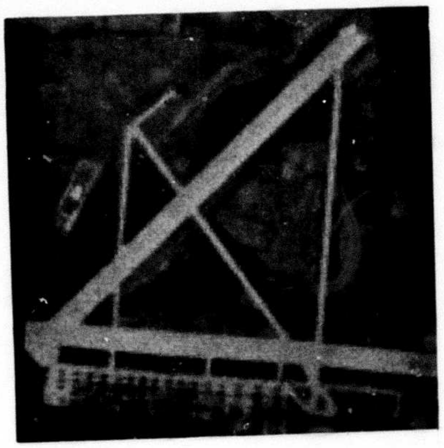
In this report, the problem of combining collinear antiparallels [1] in order to detect elongated objects, such as roads and runways, in an aerial image is considered. The problem presents itself when an aerial image is processed by a general linear feature extraction program such as [1]: the program breaks the straight boundaries into several disjoint lines.

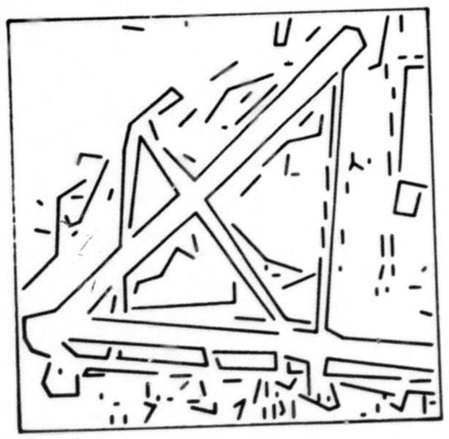
An image of an airport area is used as an example to illustrate the algorithm's effectiveness. Fig. 1(d) shows the results obtained by processing the antiparallel data of Fig. 1(c) with the algorithm to be presented. The lines represent the central axes of the runways.

Important Considerations

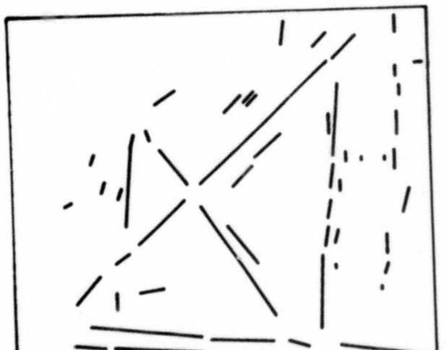
It is obvious from the nature of the problem, that we have to produce a program that will compute, or group together, collinear antiparallels in their correct spatial order. We now present three different solutions towards this end; each of them differs from the others in the way in which the antiparallels are treated. This, in turn, affects the quality of the solution which can be judged on the following three criteria:

1. The time complexity of the algorithm. (Ideally, the entire algorithm should be analyzed; however, it is not always easy to do so analytically; thus, a significant portion of the algorithm, viz., only the number of antiparallel pairs considered for collinearity evaluation, is considered in this presentation).

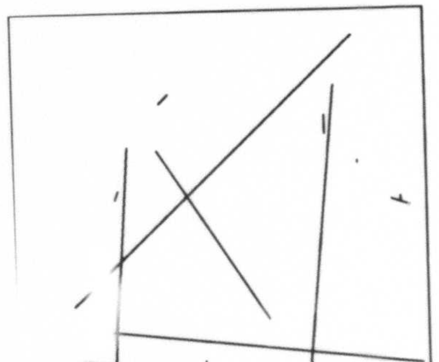




(b) Linear features.



(c) Antiparallels.



(d) Runways (only bright ones shown).

Figure 1. Runway Detection.

2. Extraneous antiparallels that do not represent elongated objects. Certain antiparallels have to be discarded because both of the line segments of such antiparallels do not have collinear continuity. In fig. 2, the line segment A does not have a collinear counterpart. In most such situations, the line represents an irrelevant detail surrounding an elongated object being sought; thus, in fig. 2, antiparallel AB can be discarded from any further consideration. This problem of selecting or discarding certain antiparallels can be called the antiparallel selection problem (APSP).

3. Computation of superfluous collinear antiparallel pairs. For example, in fig. 3, the following set of collinear antiparallels would be computed if all antiparallel pairs are compared:

$$\{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}.$$

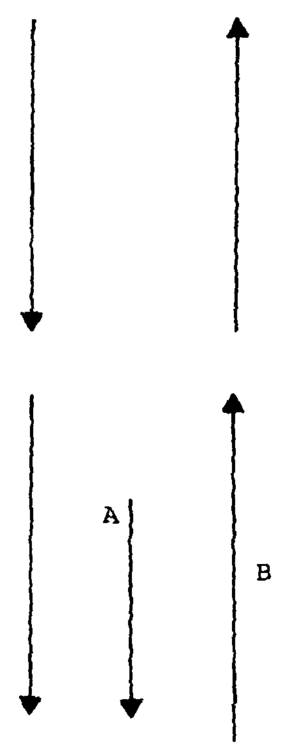
What is sufficient for characterizing the entire elongated object is, however, is only

$$\{(1,2),(2,3),(3,4)\}.$$

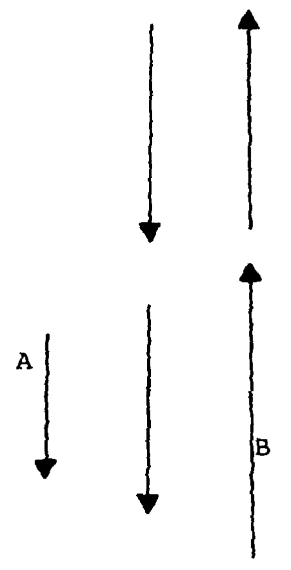
Thus, additional processing to remove this superfluity is required.

The solutions assume that the im-ge has n anciparallels.

An important aspect of a program which tries to locate collinear lines in an image is a procedure which returns a predicate indicating whether two given lines are (approximately collinear). While many variations are clearly possible, the following is a SAIL-like [2] description of what has been used to produce the results of this report:



(a)



(b)

Figure 2. The antiparallel selection problem (APSP): in both the figures, the line segment, marked A, does not form part of the runway (Thus, antiparallel AB has to be discarded).

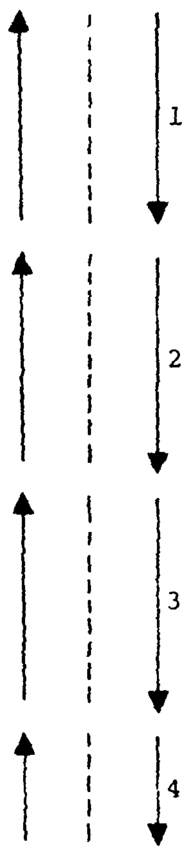


Figure 3. A hypothetical elongated object in terms of its linear features. Dotted lines represent axes of antiparallels.

```
define  colltol = " ";
  " a suitable integer for collinearity tolerance"

simple  boolean  procedure  collinear(integer dl, d2);
begin  "collinear"
" dl and d2 are the angles of vectors being tested for
  collinearity. The vector represented by d2 follows that
  represented by dl. "
boolean  ok;  integer  d;
  ok := d2 lies between dl+colltol and dl-colltol;
  if  ok  then
  begin
    d := angle made by the vector joining the tail of  1
         to the head of  2;  " angle in 0..359 "
    ok := d lies between dl+colltol and dl-colltol AND
          d lies between dl+colltol and dl-colltol;
  end;
  return(ok);
end  "collinear" ;
```

Solution #1

A straightforward algorithm is one where pairwise comparison, for collinearity, of all the antiparallels is performed. This process forms several collinear antiparallel pairs; some more processing is required to group the (collinear) antiparallels representing a single elongated object. The characteristics of this algorithm are -

1. Complexity of comparisons is O(n^2).

2. APSP can be solved only by conducting a search among collinear antiparallel pairs (cf. Solution #3).

3. A considerable number of superfluous collinear antiparallel pairs is computed only to be discarded later.

Solution #2

Instead of comparing all antiparallel pairs for possible collinearity, it is possible to sort the antiparallels into buckets representing their angular orientation. Then, antiparallels in neighboring buckets only need be considered for the collinearity comparison. The characteristics of this algorithm are -

1. The complexity here is not reduced, compared to that in solution #1. If we assume uniform distribution of antiparallels into buckets, the number of antiparallels in each bucket is proportional to n, and since collinears for n antiparallels have to be computed, the complexity of comparisons is $O(n^2)$, possibly with a reduced constant.

2. Same as in solution #1.

3. Same as in solution #1.

Solution #3

Most of the inefficiency in the above two solutions is due to the consideration, for collinearity checking, of antiparallel pairs which are not proximate; thus, we are led, in this solution, to exploring an algorithm which will enable the program to consider proximate antiparallels only.

Such a consideration is easily achieved by a point representation of the antiparallels on an image grid (Fig. 4). Each antiparallel is represented by two point vectors, each positioned at one of the two endpoints and pointing outward (of the enclosed rectangle of the

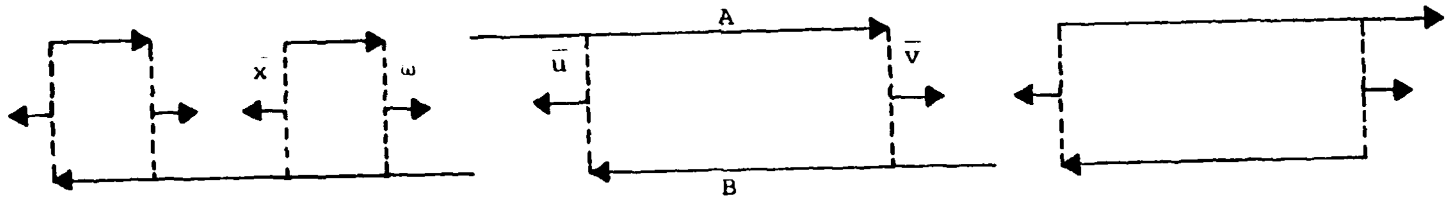


Figure 4. Representation of antiparallels: the point vectors $\bar{u}$ and $\bar{v}$ represent the antiparallel made by line segments A and B. The point vectors $\bar{\omega}$ and $\bar{u}$ "face" each other; but neither $\bar{u}$ and $\bar{v}$, nor $\bar{\omega}$ and $\bar{x}$.

antiparallel) and parallel to the antiparallel. (In some situations, two endpoints may have the same image coordinates; in that case, relocating one of them a pixel or two away will be sufficient to preserve their identity while, at the same time, not significantly affect the relative positioning of the antiparallels).

The essence of the algorithm is an operation which scans each potential neighborhood for compatible point vectors. This compatibility of point vectors captures collinear antiparallels and is defined by -

1. the point vectors representing the antiparallels are <u>anticollinear</u>, i.e., the angles differ by $180 \pm \alpha$, a tolerance, and, the point vectors (Fig. 4), and

2. the component segments of the antiparallels form two collinear pairs. Each application of the operator will result in at most one antiparallel pair.

This operation over the entire image produces a list of non-redundant collinear antiparallel pairs. These pairs are then examined to produce lists of pairs such that if the pairs (a_1,b_1), $(a_2,b_2),\ldots,(a_n,b_n)$ are in the list, then $b_i=a_{i-1}$, $i=2,\ldots,n-1$. Each list, then, represents an elongated object or, equivalently, a collection of collinear antiparallels.

The characteristics of this algorithm are -

1. Since the operation scans only around a point vector (representing an antiparallel), and since the maximum number of point vectors that can come under the purview of the scan area is a constant (determined by the area covered by the operation), the number of collinearity comparisons is $O(n)$.

2. Since for each point vector, only one compatible point vector can

exist, the extraneous antiparallels are never computed.

3. Because of proximity considerations inherent in the operation, no superfluous collinear antiparallel pairs are computed.

References

[1]. R. Nevatia and K. Ramesh Babu, "Linear Feature Extraction and Description," Computer Graphics and Image Processing, Vol. 13, No. 3, June 1980, pp. 257-269.

[2]. J. F. Reiser, "SAIL," Computer Science Department, Report STAN-CS-76-574, August 1976.

2.7 Shape Matching Using Hierarchical Gradient Relaxation Technique

B. Bhanu and O.D. Faugeras

Introduction

The problem of shape matching mainly consists of two parts [1].

1) One shape is recognized to be an approximate match to another shape, and

2) A piece of a shape is recognized as an approximate match to a part of a larger shape. This problem is also known as the "segment matching" [2].

In this paper we address the "segment matching" problem of shape matching using hierarchical gradient relaxation technique.

The class of shapes that we consider are represented by simple closed curves (no holes) and are two dimensional in nature such as the linear features in satellite images, borders on maps, outlines of biological cells etc.

One of the earliest shape matching techniques is based on chain code cross-correlation [3]. However, the usefulness of this method as a solution to the segment matching problem is very much limited by the fact that cross-correlation is not rotation invariant, it is very sensitive to local changes in the number of chain links and also quite sensitive to small global changes in the shape. There exist a large number of statistical pattern recognition techniques for shape matching [4]. Unfortunately, these feature based approaches cannot be used for segment matching problem because they suffer from the fact that the descriptors of a segment of a shape do not ordinarily bear any simple relationship to the descriptors of the entire shape. Another class of shape matching techniques are syntactic techniques [5], but they have not been applied to the segment matching problem. Davis [1,6] studies segment matching using discrete relaxation methods which carry strong syntactic flavor. Davis and Rosenfeld [7,8] use iterative methods for recognizing upright squares on a noisy background and hierarchical relaxation for waveform parsing. However, they do not define a hierarchical relaxation network and study its usefulness and computational properties [1]. Rutkowski [9,10] considers the shape segmentation of closed boundary curves such as aeroplane using relaxation methods. Kitchen [11,12] applies discrete and fuzzy logic approaches of relaxation for matching relational structures. In fact, relaxation methods have been applied to a wide variety of problems in pattern recognition, scene analysis and artificial intelligence. A good survey of these methods is given in [13]. Rosenfeld, Hummel and Zucker [14] introduced the idea of probabilistic relaxation and Faugeras and Berthod [15] reformulated the probabilistic relaxation problem by explicitly minimizing a criterion function. In this paper we follow this optimization approach of relaxation, called the gradient relaxation method. This method has been applied to segmentation, semantic description of aerial pictures, edge detection etc. [15,16,17]. We maximize a criterion function using projection gradient method and solve the shape matching problem in a hierarchical manner.

In this contribution we consider only two levels of hierarchy and show that how the method can be generalized. Various ways of computing the initial probabilities and compatibilities are described. Interaction of the two levels of relaxation is explained and strategies that lead to faster computation are introduced. Finally, an example is discussed in detail.

Shape Matching Using Gradient Relaxation Technique

In this section we present a two stage hierarchical gradient relaxation method (fig. 1) for matching the segments of a template against the segments of an object. Let $O=(O_0,O_1,...,O_{L-2})$ and $T=(T_0,T_1,...,T_{N-1})$ be the polygonal path representation of the object and the template respectively. The segments are the sides of the polygon and conventionally a polygon will be traced in the clockwise sense. In general L may be greater, equal or less than N.

In the following discussion template elements will be called as units and object elements as classes. Since a unit may not correspond to any of the object elements, there exists a $(L-1)^{th}$ class, known as the nil class and denoted by O_{L-1}. So we have N units and L classes. Relaxation is an iterative process for labeling a set of interrelated units. In "probabilistic relaxation" discussed below, a set of estimates of class assignment probabilities is initially associated with each unit. At subsequent iterations, the probabilities are adjusted in accordance with the support they receive from the class probabilities of related units.

To each of the units T_i, we assign a probability denoted by $p_i(k)$ to belong to class O_k. This is conveniently represented as a vector

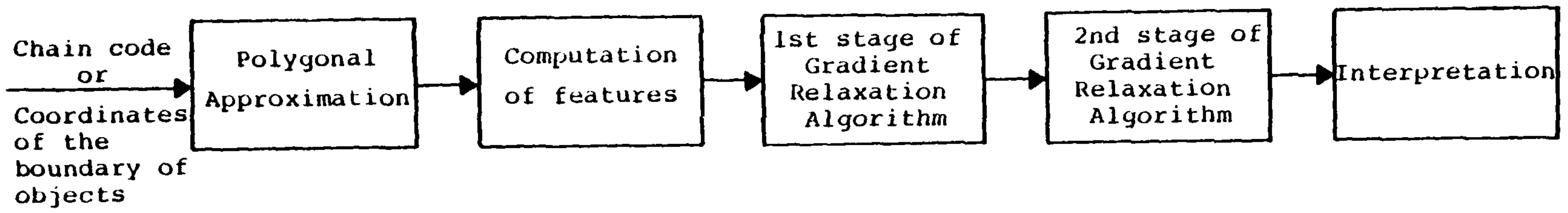

Chain code
or
Coordinates of the boundary of objects
Polygonal Approximation
Computation of features
1st stage of Gradient Relaxation Algorithm
2nd stage of Gradient Relaxation Algorithm
Interpretation

Fig. 1. Block diagram of shape matching algorithm using 2 stages of Hierarchical Gradient Relaxation Algorithm.

$\vec{p}_i=[p_i(0),\ p_i(1),\ldots,p_i(L-2),\ p_i(L-1)]^T$. The set of all vectors $p_i(i=0,\ldots,N-1)$ is called a probabilistic or stochastic labeling of the set of units. Units are related to one another, set of units related to T_i is denoted by V_i. The units that are related to T_i are $T_{((i-1))_N}$ and $T_{((i+1))_N}$, where the indices are taken as modulo N, the number of units. (For the sake of notational simplicity, in the rest of the paper we shall not use the modulo notation explicitly). At the first stage of hierarchy T_{i-1} and T_{i+1} will be called as the left and right neighbors of the unit T_i. At the second stage of hierarchy T_{i-1}, T_i, and T_{i+1} will be considered as an entity in itself. The world model is specified by the compatibility function C, which in general is defined only over a subset $S_1 \subseteq (N\text{x}L)^2$ for the first stage and $S_2 \subseteq (N\text{x}L)^3$ for the second stage of hierarchy. At the first stage of hierarchy $C(T_i,O_k,T_j,O_\ell)$ measures the adequacy of calling unit T_i as O_k and unit T_j as O_ℓ where $T_j \in V_i(=T_{i-1}$ or $T_{i+1})$. Similarly at the second stage of hierarchy $C(T_i,\ O_k,\ T_{i-1},\ O_{\ell_1},\ T_{i+1},\ O_{\ell_2})$ measures the adequacy of calling unit T_i as O_k, unit T_{i-1} as O_{ℓ_1} and unit T_{i+1} as O_{ℓ_2}. For each of the units we also define a consistency vector $\vec{q}_i=[q_i(0),\ q_i(1),\ldots,q_i(L-2),\ q_i(L-1)]^T$ that tells us what $\vec{p}_i$ should be given $\vec{p}_j$ at the neighboring related units and the compatibility function. For simplicity in the sequel we shall denote $C(T_i,\ O_k,\ T_j,\ O_\ell)$ as $C(i,k,j,\ell)$ and $C(T_i,\ O_k,\ T_{i-1},\ O_{\ell_1},\ T_{i+1},\ O_{\ell_2})$ as $C(i,\ k,\ i_1,\ \ell_1,\ i_2,\ell_2)$.

As described in [15], we define

$$q_i(k) = \frac{Q_i(k)}{\sum_{\ell=0}^{L-1} Q_i(\ell)} \tag{1}$$

where,

$$Q_i(k) = \sum_{j \varepsilon V_i} \sum_{\ell=0}^{L-1} C(i,k,j,\ell) p_j(\ell) \qquad (2)$$

at the first stage of hierarchy. Similarly for the second stage of hierarchy we obtain,

$$Q_i(k) = \sum_{\ell_1=0}^{L-1} \sum_{\ell_2=0}^{L-1} C(i,k,i_1,\ell_1,i_2,\ell_2) p_{i_1}(\ell_1) p_{i_2}(\ell_2) \qquad (3)$$

with $q_i(k)$ defined by (1). The global criterion that measures the consistency and ambiguity of the labeling over the set of units is given by

$$c = \sum_{i=0}^{N-1} \vec{p}_i \cdot \vec{q}_i \qquad (4)$$

we carry out the maximization of (4) using the projection gradient approach. This criterion has been successfully used in the segmentation and semantic description of aerial images [16,17]. The labeling problem now becomes to find the local maximum of the criterion c closest to the original labeling $\vec{p}_i^{(0)}$ subject to the constraint that $\vec{p}_i$'s are probability vectors. This problem can be efficiently solved using steepest descent techniques [15]. The gradient of the criterion c is obtained as,

$$\frac{\partial C}{\partial p_i(k)} = q_i(k) + \sum_{j \varepsilon V_i} \frac{1}{D_j} \sum_{\ell=0}^{L-1} C(j,\ell,i,k) \cdot (p_j(\ell) - \vec{p}_j \cdot \vec{q}_j) \text{ for } k=0,1,\ldots,L-1$$

$$\text{where} \quad D_j = \sum_{\ell=0}^{L-1} Q_j(\ell) \tag{5}$$

at the first stage of hierarchy. The first term in (5) corresponds to the simple maximization of the product $\vec{p}_i \cdot \vec{q}_i$ in the global criterion c, and the second term corresponds to the coupling between units through the compatibility function. Note that in general $C(j,\ell,i,k) \neq C(i,k,j,\ell)$ since it depends upon the manner in which the compatibility is computed. More about this is explained under the compatibility computation.

Similarly, at the second stage of hierarchy the gradient of the criterion c is obtained as,

$$\frac{\partial C}{\partial p_i(k)} = q_i(k) + \frac{1}{D_{i_1}} [\tilde{Q}_{i_1}(k) - D'_{i_1} \vec{p}_{i_1} \cdot \vec{q}_{i_1}] + \frac{1}{D_{i_2}} [\tilde{Q}_{i_2}(k) - D'_{i_2} \vec{p}_{i_2} \cdot \vec{q}_{i_2}], \tag{6}$$

where,

$$\tilde{Q}_{i_1}(k) = \sum_{\ell_1=0}^{L-1} \sum_{\ell_3=0}^{L-1} C(i_1,\ell_1,i_3,\ell_3,i,k) p_{i_1}(\ell_1) p_{i_3}(\ell_3)$$

$$\tilde{Q}_{i_2}(k) = \sum_{\ell_2=0}^{L-1} \sum_{\ell_4=0}^{L-1} C(i_2,\ell_2,i_4,\ell_4,i,k) p_{i_2}(\ell_2) p_{i_4}(\ell_4)$$

$$D'_{i_1} = \frac{\partial D_{i_1}}{\partial p_i(k)} = \sum_{\ell_1=0}^{L-1} \sum_{\ell_3=0}^{L-1} C(i_1,\ell_1,i_3,\ell_3,i,k)\cdot p_{i_3}(\ell_3)$$

$$D'_{i_2} = \frac{\partial D_{i_2}}{\partial p_i(k)} = \sum_{\ell_2=0}^{L-1} \cdot\sum_{\ell_4=0}^{L-1} C(i_2,\ell_2,i_4,\ell_4,i,k) p_{i_4}(\ell_4)$$

$$D_{i_1} = \sum_{\ell=0}^{L-1} Q_{i_1}(\ell)$$

and

$$D_{i_2} = \sum_{\ell=0}^{L-1} Q_{i_2}(\ell)$$

These notations are made clear in Fig. 2. Again it is to be noted that compatibilities like $C(i_1,\ell_1,i_3,\ell_3,i,K)$ need not be equal to $C(i,k,i_i,\ell_i,i_3,\ell_3)$, since it depends upon the manner of computation. Now the iteration of p_i's is given by,

$$p_i^{(n+1)}(k) = p_i^{(n)}(k)+\rho_i^{(n)} P_i^{(n)}\left[\frac{\partial C}{\partial p_i(k)}\right] \tag{7}$$

for $k=0,1,\ldots,L-1$ and $i=0,1,\ldots,N-1$.

where,

$$P_i^{(n)}\left[\frac{\partial C}{\partial p_i(k)}\right] = \frac{\partial C}{\partial p_i(k)} - \frac{1}{L}\sum_{K=0}^{L-1} \frac{\partial C}{\partial p_i(k)} = C_1 \tag{8}$$

Normally, $\rho_i^{(n)}$ is kept constant for all units during each iteration

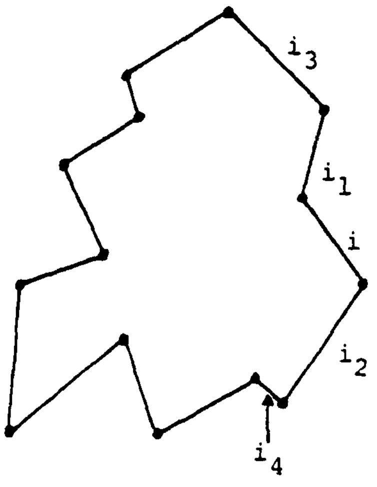


Fig. 2. Explanation of the notations for (6).

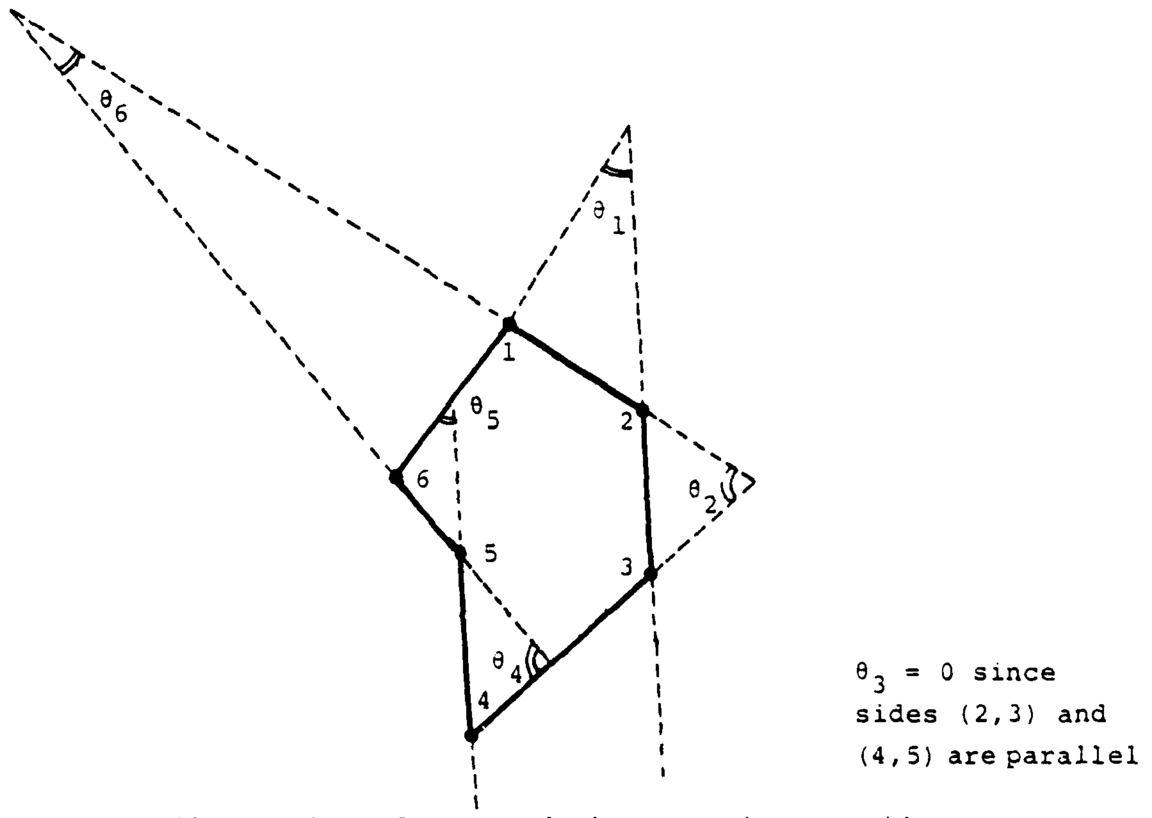


Fig. 3. Illustration of the angle between the two sides.

and is determined to have the largest possible value such that p_i's at the (n+1)th iteration still lie in the bounded convex region of LN dimensional Euclidean space defined by $\sum_{k=0}^{L-1} p_i(k)=1$ and $p_i(k)\geq 0$, $i=0,\ldots,N-1$. However, to obtain the faster convergence rate $\rho_i^{(n)}$ is obtained as

$$\rho_i^{(n)} = \alpha \cdot \min[\max_K \rho_i^{(n)}(k)] \qquad (9)$$

where α is a constant between 0 and 1 and can be used to control the rate of convergence. Also

$$\rho_i^n(k) = \frac{1-p_i^{(n)}(k)}{C_1}, \text{ if } C_1 > 0 \qquad (10)$$

$$= \frac{p_i^{(n)}(k)}{C_1}, \quad \text{if } C_1 < 0$$

A side effect of computing $\rho_i^{(n)}$ for every unit is that we may not be following the gradient exactly. However, it can be expected that we are approximately in the direction of the gradient and the criterion (4) is still maximized. Although the criterion generally increases but sometimes it decreases slightly because $\rho_i^{(n)}$ in (9) is too large for some of the units and then it continually increases.

Details of the Shape Matching Algorithm

In this section we study details of the shape matching algorithm whose block diagram is shown in fig. 1.

Polygonal Approximation

Polygonal approximation for the template and object is found by using the algorithm proposed by Rosenfeld and Johnston [18], which detects the points of high curvature. The algorithm works as follows.

Let $R=\{(X_i,Y_i)\}_{i=1}^{n}$ be the sequence of points describing a closed curve so that $(X_1,Y_1)=(X_n,Y_n)$. At every point of the sequence, smoothed k-curvature is evaluated as,

$$C_{ik} = (\vec{a}_{ik} \cdot \vec{b}_{ik}) / |\vec{a}_{ik}| \cdot |\vec{b}_{ik}|$$

where

$$\vec{a}_{ik} = (X_i - X_{i+k}, Y_i - Y_{i+k})$$

$$\vec{b}_{ik} = (X_i - X_{i-k}, Y_i - Y_{i-k})$$

C_{ik} is the cosine of the angle between the vectors $\vec{a}_{ik}$ and $\vec{b}_{ik}$ so that $-1 \leq C_{ik} \leq 1$, and $C_{ik}=-1$ for a straight line and +1 for the sharpest angle of 0 degrees. Thus, the local maxima of C_{ik} or the local minima of the angle will correspond to the points of maximum curvature, which will serve as the vertices of the polygonal approximation. Although this method has certain shortcomings [19], but it works well if the largest k is smaller than the distances between successive angles along the curve. Normally smoothing factor k is taken about 1/5 or 1/10 of the perimeter.

Features Derived From Polygonal Approximation

Some of the features that can be derived from polygonal approximation of the boundaries of an object are-

1) Length of a side.
2) Intervertices distance.
3) Slope of a side.
4) Interior angle between the two sides, and
5) Angle between the two sides as shown in fig. 3. This angle corresponding to a vertex is equal to the angle between the two sides, where one side is obtained by joining this vertex and its neighboring counter clockwise vertex and the other side is obtained by joining the two clockwise neighboring vertices of this vertex. This angle is

named as exangle.

When scale invariant features are derived, slope, interior and exangle features can be used. For rotation invariance length of a side and intervertices distance can be used. For rotation as well as scale invariance interior or exangle or a combination of angle and length features can be used in the initial assignment of probabilities.

<u>Initial Assignment of Probabilities</u>

The initial assignment of probabilities for a unit is obtained by comparing its feature values with the feature values of all the object segments. Depending upon the type of invariance desired, we may need the weight and the strength of a particular feature. In general, the quality of correspondence of a unit i to an object segment k is given by,

$$M(T_i,O_k) = \sum_{n_1=1}^{N1} |f_{tn_1} - f_{on_1}| \, W_{n_1} S_{n_1} \tag{11}$$

where N1 is the total number of features,

f_{tn_1}=n_1 feature value for the template element,
f_{On_1}=n_1 feature value for the object element,
W_{n_1}=weight for the feature n_1, and
S_{n_1}=strength associated with the feature n_1.

Note that for a perfect match $M(T_i,O_k)$=1 and for a poor match $M(T_i,O_k)$ will be very small. Thus the initial probability is assigned as,

$$p_i(k) = \frac{1}{1+M(T_i,O_k)}, \quad k=0,1,\ldots,L-2$$

These probabilities are normalized so that they sum to 1. If we use only either scale invariance or rotation invariance or both, we may not need weight and strength factors since only one type of features (N1=1) are involved. However, if we combine length and angles, then we need the strength and weight to account for their importance and the different range of values of features. Thus the initial assignment of probabilities involve unary relations.

<u>Computation of Compatibilities</u>

The compatibility function determines the degree by which two or three assignment of the units are compatible with each other. There are at least 4 ways of computing the compatibilities at the first and second stage of hierarchy.

<u>First Method</u>

At the first stage, we want a transformation TR from unit T_i to label O_k, i.e., TR: $T_i \rightarrow O_k$. TR consists of scale, rotation and translation in the X and Y directions. This transformation is applied to the unit T_j and the error between the transformed T_j and O_ℓ is computed as,

$$M(TR(T_j),O_\ell) = \sum_{n_1=1}^{N1} |f_{t'n_1} - f_{on_1}| \, W_{n_1} S_{n_1} \qquad (12)$$

where $f_{t'n_1} = n_1$ feature value for the transformed unit, and the other quantities are similar to those in (11).

Note that here the features may be slope and length of the side, so we shall need the weight and strength for these features, if the matching error is based on these features. However, it is possible to avoid these parameters, if we use only the distance between the ends of O_ℓ and transformed T_j as the matching error between two segments. As illustrated in fig. 4 in this case matching error

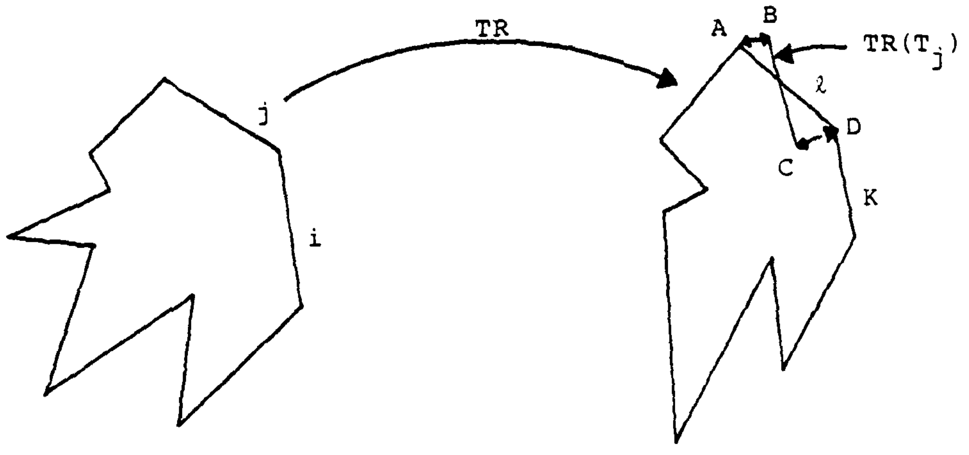


Fig. 4. Matching distance error = AB + CD.

$M(TR(T_j),O_\ell)=AB+CD$. Now the compatibility

$$C(i,k,j,\ell) = \frac{1}{1+M(TR(T_j),O_\ell)}$$

The problem with this method of computing the compatibilities is that the compatibilities are not symmetric i.e., $C(i,k,j,\ell) \neq C(j,\ell,i,k)$. As we have seen the computations of gradient requires $C(j,\ell,i,k)$, so if this method is used we will need some modification in our program. Moreover, since we are using only one transformation, compatibilities values will not be very accurate compared to the other three methods described below.

At the second stage to compute $C(i,k,i_1,\ell_1,i_2,\ell_2)$ still we find one transformation as described above and use it to compute the error between the transformed i_1 and ℓ_1 (Error 1) and transformed i_2 and ℓ_2 (Error 2). Then

$$C(i,k,i_1,\ell_1,i_2,\ell_2) = \frac{1}{1+\text{Error 1}+\text{Error 2}}$$

Because of the problems of asymmetry and inaccuracy, this method is not much used.

<u>Second Method</u>

Unlike the first method, here we find two transformations TR1 and TR2 such that

$$TR1: \quad T_i \rightarrow O_k$$

and,

$$TR2: \quad T_j \rightarrow O_\ell$$

Now the average rotation, average scale and average translation (in the X and Y directions) of these two transformations is computed. The transformation associated with these parameters called TV, is now applied to unit i and unit j and the matching errors between the transformed units and the elements O_k and O_ℓ are computed as in the first method and finally,

$$C(i,k,j,\ell) = \frac{1}{1+\text{Total Error}}$$

At the second stage instead of finding two transformation, we find three transformations and take the average of these values. This average transformation is then applied to units i, i_1, i_2 and the total error between the transformed units and object elements k, ℓ_1, ℓ_2 is computed to get $C(i, k, i_1, \ell_1, i_2, \ell_2)$ as in the first stage.

Third Method

This method is similar to the second method in that we compute two transformations TR1 and TR2. Now TR1 is applied to T_j giving matching error $M(TR1(T_j),O_\ell)$ and TR2 is applied to T_i giving matching error $M(TR2(T_i),O_\ell)$. Average of this error is taken and

$$C(i,k,j,\ell) = \frac{1}{1+\text{Average Error}}$$

At the second stage, we will find three transformations and the average error will be the average of six error terms and the compatibility

$$C(i,k,i_1,\ell_1,i_2,\ell_2) = \frac{1}{1+\text{Average Error}}$$

Fourth Method

In this method we compute mathematically the best transformation from units i and j such that the sum of the squares of the error between the transformed units and the object elements is minimum. Here we can use only distance for the computation of matching error (unlike the first three methods where we could use a combination of slope and length) so that the error criterion is linear and linear least squares ideas can be applied. An example is given below. Let the coordinates of segments i,j,k, and ℓ be given by (X_1,Y_1), (X_2,Y_2), (DX_1,DY_1), (DX_2,DY_2), (R1,S1), (R2,S2), (U1,V1) and (U2,V2) respectively then

$$MX=b$$

where

$$M = \begin{bmatrix} U1 & -V1 & 1 & 0 \\ V1 & U1 & 0 & 1 \\ U2 & -V2 & 1 & 0 \\ V2 & U2 & 0 & 1 \\ R1 & -S1 & 1 & 0 \\ S1 & R1 & 0 & 1 \\ R2 & -S2 & 1 & 0 \\ S2 & R2 & 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} \lambda\cos\theta \\ \lambda\sin\theta \\ X_0 \\ Y_0 \end{bmatrix}, \quad b = \begin{bmatrix} DX1 \\ DY1 \\ DX2 \\ DY2 \\ X1 \\ Y1 \\ X2 \\ Y2 \end{bmatrix}$$

and λ is a scaling factor, θ is the rotation and X_0 and Y_0 are the translations in the X and Y directions respectively. The above set of equations is an overdetermined system. It can be transformed as

$$M^TMX=M^Tb$$

and now it will be 4x4 system, which can be uniquely solved for λ, θ, X_0, Y_0. This computed transformation can then be applied to obtain compatibilities similar to the second method.

At the second stage we will have M as 12x4 matrix and b a column vector of size 12x1. It can be solved exactly as above to obtain the compatibilities. This method requires more time for the computation of compatibilities, than any of the other methods. Also note that the second, third and fourth methods of compatibility computation lead to symmetric compatibilities.

<u>Initial Probability and Compatibility for the NIL Class</u>

The assignment of initial probability and compatibility to the nil class is very important, since for some of the units there may not be any object element. Note that O_{L-1} is the nil class.

<u>Assignment of Initial Probability to the Nil Class p_i(nil)</u>

There are at least three ways for the initial assignment of the probability to the nil class.

1. All the classes are assumed to be equally likely, then

$$p_i(\text{nil}) = p_i(0) = p_i(1) \ldots = p_i(L-2) = 1/L;$$

where L is the total number of classes. However, it is not a good assignment since the results of relaxation scheme depends on the initial assignment and it is better to use feature information for classes (other than nil class) rather than using no information.

2. p_i(nil) is assigned a small constant value, depending upon the a priori information that we may have about the possible number of matches. Normally, we have taken p_i(nil) between 0.05 to 0.25. The actual value is not critical, however, it affects the convergence of probabilities, hence the number of iterations required to achieve the desired result.

3. If the number of units is not very large, then p_i(nil) can be taken as

$$p_i(\text{nil}) = 1 - \max_k p_i(k), \qquad k = 0,1,\ldots,L-2$$

After the assignment of probability to the nil class in the above two cases probabilities are again normalized so that they sum to 1 for L classes.

Assignment of Compatibilities Involving Nil Class

At the first stage of hierarchy the compatibility is $C(i,k,j,\ell)$. Compatibility involving nil class is assigned as follows. (Note that q_{L-1} is the nil class).

$$C(i,k,j,nil) = p_i(k)$$

$C(i,nil,j,\ell)$=small number usually between 0.05 and 0.25 where, ℓ varies over all classes.

At the second stage of hierarchy the compatibility is given by $C(i,k,i_1,\ell_1,i_2,\ell_2)$. Compatibilities involving nil classes are assigned as follows.

$$C(i,k,i_1,nil,i_2,\ell_2) = C(i,k,i_2,\ell_2)$$
$$C(i,k,i_1,\ell_1,i_2,nil) = C(i,k,i_1,\ell_1)$$
$$C(i,k,i_1,nil,i_2,nil) = p_i(k)$$
$$C(i,nil,i_1,\ell_1,i_2,\ell_2) = \text{small no. usually between 0.05 and 0.25}$$

where ℓ_1 and ℓ_2 vary over all classes.

Strategies That Lead to Faster Computation

Following are some of the strategies that have been used to obtain faster convergence of the probabilities.

1) We can threshold a probability value to zero if it is less than a certain threshold such as 10^{-4}. Once one or more of the components of

$\vec{p}_i$ become zero, we don't compute the gradient and compatibility for them and suitably take care of it in the computation of the projection of the gradient.

2) We can threshold a probability vector $\vec{p}_i$ as the unit vector if any of the components of $\vec{p}_i$ becomes greater than a certain threshold, say 75%.

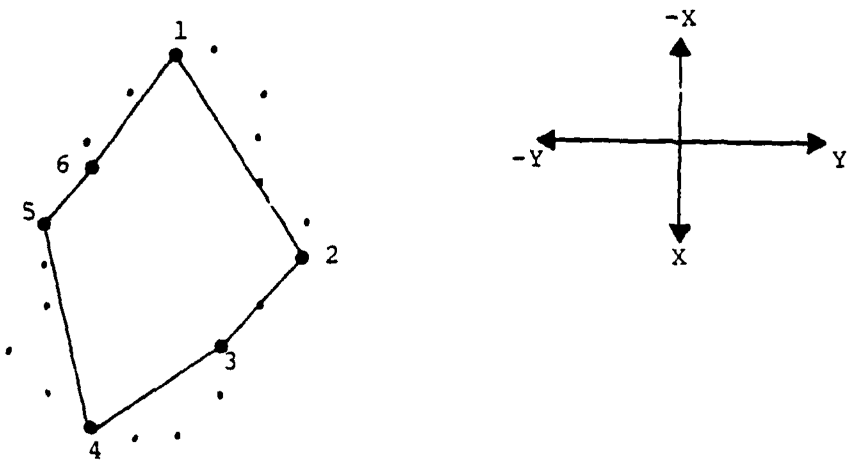
3) Compute $Q_i(k)$, compatibility and gradient at the first and second stage only for a limited number of most likely labels. Usually this number has been taken to be 1 or 2.

All of these features have been incorporated in the program and lead to a marked reduction in the computation time. In the next section we present an example which illustrates the capabilities of hierarchical gradient relaxation method.

An Example

Figs. 5 and 6 show a template and an object respectively. The perimeter of the template consists of 21 points and that of the object 38 points. Various features of the template and object are shown in these figures. V_X and V_Y correspond to the X and Y coordinates of the vertices. Polygonal approximation is obtained with a smoothing factor of 4. It is to be noted that the upper portion of the template is a noisy version of the object so the polygonal approximation in the two cases are different. This makes the matching problem little more complicated than in [1] where Davis introduces the noise after the polygonal approximation, so that the number of edges and vertices remain the same.

Table 1 and 2 show the results for the two levels of hierarchy (8 first stage iterations followed by 8 second stage iterations) at various iterations and table 3 shows the expected assignments for the units of the template. Initial probabilities have been computed using



Smoothing factor = 4

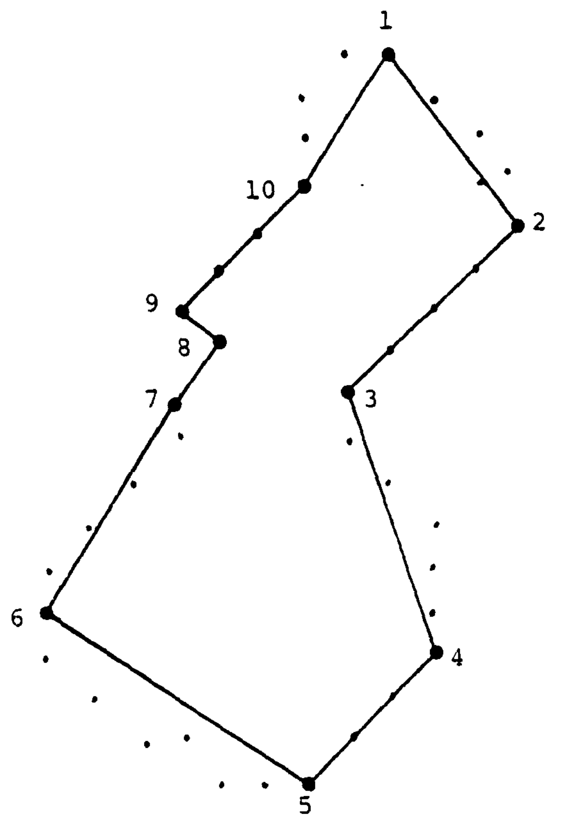
# of vertices = 6

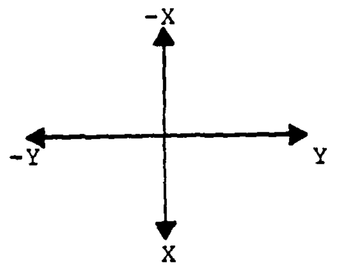
Perimeter of the template = 21

Features of the template

V_X, V_Y	Vertices	Slope	Length	Interior angle	Exangle
21,22	1	30.96	5.83	65.0	11.30
26,25	2	315.00	2.82	104.0	87.27
28,23	3	303.69	3.60	169.0	56.30
30,20	4	191.30	5.09	67.0	11.30
25,19	5	135.00	1.41	124.0	45.00
24.20	6	146.30	3.60	191.0	75.96

Fig. 5. Template.





Smoothing factor = 4
# of vertices = 10
Perimeter of the object = 38

<u>Features of the object</u>

V_X, V_Y	Vertices	Slope	Length	Interior angle	Exangle
22,24	1	36.86	5.00	71.0	11.30
26,47	2	315.00	5.65	98.0	18.43
30,43	3	18.43	6.32	243.0	0.00
36,45	4	315.00	4.24	117.0	37.87
39,42	5	236.30	7.21	101.0	14.03
35,36	6	149.03	5.83	93.0	78.69
30,39	7	135.00	1.41	166.0	75.96
29,40	8	225.00	1.41	270.0	0.00
28,39	9	135.00	4.24	90.0	78.69
25,42	10	146.30	3.60	191.0	81.86

Fig. 6. Object.

Table 1. Reduced compatibility and gradient computation.

Smoothing factor = 4, α = 0.99, nil class value = 0.15

Type of compatibility computation = 3

Unit	Assigned class at various iterations																
	First Stage of Hierarchy									Second Stage of Hierarchy							
	0	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	5	5	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	7	7	7	7	7	7	7	7	7	7	7	7	7	7	2	2	2
4	1	1	1	1	1	1	1	1	1	11	11	11	11	11	11	11	11
5	4	4	4	11	11	11	9	9	9	11	11	11	11	11	9	9	9
6	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10

Total computation time = 19.39 seconds.

Table 2. Full compatibility and gradient computation.

Smoothing factor = 4, α = 0.99, nil class value = 0.15

Type of compatibility computation = 3

Unit	Assigned class at various iterations																
	First Stage of Hierarchy									Second Stage of Hierarchy							
	0	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	5	5	5	5	2	2	2	2	2	2	2	2	2	2	2	2	2
3	7	7	7	7	7	7	7	7	7	7	7	11	2	2	2	2	2
4	1	1	1	1	1	1	1	1	1	1	1	8	8	9	9	9	9
5	4	4	4	4	4	4	4	4	4	9	9	10	10	10	10	9	9
6	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10

Total computation time = 420.35 seconds.

Table 3. Expected assignments for the units of the template.

Unit	Assigned class
1	1
2	2
3	2
4	9 or 11
5	9 or 10 or 11
6	10

only the angle between the two sides (i.e. interior angle) and compatibilities have been obtained by finding the average error (third method). In obtaining Table 1 we have considered only one most likely name in computing compatibilities and gradient, whereas in Table 2, all possible class names have been considered. Computation time for Table 1 is about 1/20 of that for Table 2. About 70% of the total computation time is spent in computing the gradient. Also since we don't store the compatibility values while computing the consistency vector, we compute them again when gradient is needed. Assignment in both cases (Tables 1, 2) are very reasonable although we did not converge to the same assignment. If we increase the number of iterations for the first stage in Tables 1 and 2, then corresponding to Table 1 at the 15th iteration of the first stage, the assignment of the units is 1, 2, 7, 8, 9, 10 and corresponding to Table 2 it is 1, 2, 2, 8, 9, 10. This shows the need for the second stage which gives a better labeling of units when it is followed by the first stage. Also we have found that generally the sum of the iterations at the 1st and 2nd stage is usually less than needed by the 1st stage alone to obtain approximate labeling of units. In obtaining these tables we have not thresholded the upper values of the probabilities (the lower values have been thresholded at 10^{-4}). However, if we assign a unit vector to a unit when its probability for a particular class gets higher than a threshold, say 60%, then the number of iterations at both stages reduces.

Conclusion

We have shown how a hierarchical gradient relaxation method can be successfully used for the shape matching of objects. In this study although we have considered only two levels of hierarchy, yet the algorithm easily generalizes to higher levels of hierarchy. With the increase in levels of hierarchy, we are using more world knowledge, so the complexity of the processing increases, but the reliability of the assignment of units increases. First stage alone does not resolve all the ambiguities of labeling even if the number of iterations is very large. Second stage helps in resolving those ambiguous labelings. Normally we have found that about L/2 number of iterations at the first stage followed by an equal number of iterations at the second stage produce reasonable matches when $L \leq 20$. Although the chain code cross-correlations and feature based approaches are simpler and computationally less costly, yet they are not as robust and flexible as this method is. Also this method does not require that the number of template elements be less than the number of object elements as it is needed in [1].

References

[1]. L.S. Davis, "Shape Matching Using Relaxation Techniques," IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No. 1, Jan. 1979, pp. 60-72.

[2]. H. Freeman, "On the encoding of arbitrary geometric configurations," IRE Trans. Electron. Computers, Vol. EC-10, 1961, pp. 260-268.

[3]. H. Freeman, "Computer Processing of line-drawing images," Computing Surveys, Vol. 6, No. 1, March 1974, pp. 57-97.

[4]. R.O. Duda and P.E. Hart, "Pattern Classification and Scene Analysis," John Wiley & Sons Inc., 1973.

[5]. T. Pavlidis, "A Review of Algorithms for Shape Analysis," Computer Graphics and Image Processing, Vol. 7, 1978, pp. 243-258.

[6]. L.S. Davis, "Hierarchical Relaxation for Shape Analysis," Proc. IEEE Comp. Soc. Conf. on Pattern Recognition and Image Processing, 1978, pp. 275-279.

[7]. L.S. Davis and A. Rosenfeld, "Application of Relaxation Labeling, 2: Spring-loaded Template Matching," Proc. 3rd Int. Joint

Conf. Pattern Recognition, pp. 591-597, 1976.

[8]. L.S. Davis and A. Rosenfeld, "Hierarchical Relaxation for Waveform Parsing," in Computer Vision Systems, Edited by A.R. Hanson and E.M. Riseman, Academic Press, 1978.

[9]. W.S. Rutkowski, S. Peleg and A. Rosenfeld, "Shape Segmentation using Relaxation," TR-762, May 1979, Computer Science Center, Univ. of Maryland.

[10]. W.S. Rutkowski, Shape Segmentation Using Relaxation, II," TR-793, July 1979, Computer Science Center, Univ. of Maryland.

[11]. L. Kitchen and A. Rosenfeld, "Discrete Relaxation for Matching Relational Structures," IEEE Trans. Systems, Man and Cybernetics, Vol. SMC-9, No. 12, Dec. 1979, pp. 869-874.

[12]. L. Kitchen, "Relaxation Applied to Matching Quantitative Relational Structures," IEEE Trans. Systems, Man and Cybernetics, Vol. SMC-10, No. 2, Feb. 1980, pp. 96-101.

[13]. L.S. Davis and A. Rosenfeld, "Cooperating Processes for Low Level Vision: A Survey," Technical Report 123, Jan. 1980, Univ. of Texas at Austin.

[14]. A. Rosenfeld, R. Hummel and S.W. Zucker, "Scene Labeling by Relaxation Operations," IEEE Trans. Systems, Man and Cybernetics, Vol. 6, 1976, pp. 420-433.

[15]. O.D. Faugeras and M. Berthod, "Improving Consistency and Reducing Ambiguity in Stochastic Labeling: an Optimization Approach," to appear in IEEE Trans. Pattern Analysis and Machine Intelligence.

[16]. B. Bhanu and O.D. Faugeras, "Segmentation of Images Having Unimodal Distributions," submitted to the IEEE Trans. Pattern

Analysis and Machine Intelligence, July 1980.

[17]. O.D. Faugeras and K.E. Price, "Semantic Description of Aerial Images Using Stochastic Labeling," Proc. Image Understanding Workshop, Computer Vision Laboratory, Univ. of Maryland, April 1980, pp. 89-94.

[18]. A. Rosenfeld and E. Johnston, "Angle Detection on Digital Curves," IEEE Trans. Computers, Sept. 1973, pp. 875-878.

[19]. L.S. Davis, "Understanding Shape: Angles and Sides," IEEE Trans. Computers, Vol. C-26, No. 3, March 1977, pp. 236-242.

2.8 Shape Description of Occluded Objects Using Coordinated Hierarchical Gradient Relaxation Method

B. Bhanu and O.D. Faugeras

Introduction

Matching of occluded objects is one of the prime capabilities of any shape analysis system. Particularly in the analysis of sequence of images occlusion problem becomes of major importance in the task of modeling a dynamic environment [1-3]. Aggarwal and Coworkers [4-6] studied the problem of modeling the image of a partially occluded object and use the derived model to track the object through partial or even complete occlusion. These authors work on simulated binary images, processing by systematically relaxing the constraints on object contours. Aggarwal and Duda [4] in their study on tracking clouds that partially occlude each other take the polygonal approximation of the shape of objects. These polygons are assumed to be rigid, however, holes in the polygons are allowed. The concept of a true vertex and a false vertex is used to detect occluding parts. Their matching approach is sequential and suboptimal. Chow and Aggarwal [5] consider planar curvilinear objects having no holes. They also assume that all the objects are known to the program before any analysis is done. Their basic concept of matching is the same as of Aggarwal and Duda [4]. They mention that their matching technique can be extended by matching the boundary of the images, however, approaches such as chain code cross-correlation etc. are not suitable especially if we are dealing with real images [7]. Martin and Aggarwal [6] represent the shape of an object by a sequence of straight lines which have been derived from the graph of total subtended angle vs arc length, ψ-s, function of the object. Their line-fitting process is the technique of iteration end-point fit [p. 338, 8] and their segment matching approach is heuristic and cumbersome. They consider the tracking of individual boundary segments which allows to decompose the contour of two overlapping object images during the tracking process into boundary parts related to the component objects. Yachida et al. [3] apply the ideas similar to Chow and Aggarwal [5] to the natural images involving the movement of fishes swimming in a vat. For occlusion problem, they consider the boundary matching in a relatively simple fashion. Roach and Aggarwal [9] consider the occlusion problem in a 3-D blocks world.

We view the occlusion problem, basically a segment matching problem which involves matching the segments of two or more actual objects with the apparent object, which is formed by the occlusion of these objects. Of course, some segments of the actual objects may not match with any of the segments of the apparent object. Also matching algorithm should not assign the same segment of the apparent object to the segments of different actual objects which are occluding. Once the matching of actual objects with the apparent object has been done, it will be a relatively simple matter to track them and carry out the motion analysis [15]. The segment matching algorithm that we use is based on the hierarchical gradient relaxation method [10]. It is an iterative algorithm based on optimization approach for labeling a set of interrelated units. In order to use this algorithm for the shape description of occluded objects, we modify this algorithm using a penalty function approach so that the same segment of the apparent object is not assigned to the segments of different actual objects.

The class of shapes that we consider are represented by simple closed curves (no holes) and are two dimensional in nature. These shapes will be approximated by polygons. Their vertices being the points of high curvature [11]. We allow objects to change in shape, but such changes should not be drastic otherwise no matching algorithm can work. Unlike the previous studies as mentioned above we also allow a change in scale so the objects in general may move, rotate, and their scale may change. If the objects are rigid, then matching will be a relatively easy task for our algorithm. In this paper we describe the algorithm and study the computational aspects associated with the hierarchical gradient relaxation and penalty function approach. Finally, the results are illustrated with the aid of an example in which two actual objects occlude to form an apparent object.

Problem Formulation

Consider a general case in which M ($\geq$ 2) actual objects occlude one another to form a single apparent object. In the following we shall call the actual objects as templates and the apparent object as object. Let a template X be represented by $X=(T_1,T_2,...,T_N)$, where N is the number of segments in the polygonal path representation of the template. Similarly, let $O=(O_1,O_2,...,O_{L-1})$ be the polygonal path representation for the object. Object has L-1 segments. Conventionally, a polygon will be traced in the clockwise sense, i.e. keep the interior to the right. We want to match the segments of the templates against the segments of the object such that the following two conditions are satisfied.

1) Any segment of two different templates *should not be assigned* to the same segment of the apparent object.

2) One or more segments of the templates which do not correspond to any of the segments of the apparent object should be assigned to the nil class, i.e., no match class.

We shall call the object segments as classes, and the template segments as units. Let the nil class be denoted by O_L. To each of the units T_i, we assign a probability denoted by $p_i(k)$ to belong to class O_k. This is conveniently represented as a vector $\vec{p}_i=[p_i(1),\ldots,p_i(L)]^T$. The set of all vectors $\vec{p}_i$ $(i=1,\ldots,N)$ is called a probabilistic or stochastic labeling of the set of units. Units are related to one another, set of units related to T_i is denoted by V_i. The units that are related to T_i are T_{i-1} and T_{i+1}, where the indices are taken as modulo N. At the first stage of hierarchy T_{i-1} and T_{i+1} will be called as the left and right neighbors of the unit T_i. At the second stage of hierarchy T_{i-1}, T_i, and T_{i+1} will be considered as an entity in itself. The world model is specified by the compatibility functions C, which in general is defined only over a subset $S_1 \subseteq (N\mathrm{x}L)^2$ for the first stage and $S_2 \subseteq (N\mathrm{x}L)^3$ for the second stage of hierarchy. At the first stage of hierarchy $C(T_i, O_k, T_j, O_\ell)$ measures the adequacy of calling unit T_i as O_k and unit T_j as O_ℓ where $T_j \in V_i$ $(=T_{i-1}$ or $T_{i+1})$. Similarly at the second stage of hierarchy $C(T_i, O_k, T_{i-1}, O_{\ell_1}, T_{i+1}, O_{\ell_2})$ measures the adequacy of calling unit T_i as O_k, unit T_{i-1} as O_{ℓ_1} and unit T_{i+1} as O_{ℓ_2}. For each of the units we also define a consistency vector $\vec{q}_i=[q_i(1), q_i(2), q_i(L)]^T$ that tells us what $\vec{p}_i$ should be given $\vec{p}_j$ at the neighboring related units and the compatibility function. For simplicity in the sequel we shall denote $C(T_i, O_k, T_j, O_\ell)$ as $C(i,k,j,\ell)$ and $C(T_i, O_k, T_{i-1}, O_{\ell_1}, T_{i+1}, O_{\ell_2})$ as $C(i, k, i_1, \ell_1, i_2, \ell_2)$.

As described in [10], we define

$$q_i(k) = \frac{Q_i(k)}{\sum_{\ell=1}^{L} Q_i(\ell)} \tag{1}$$

where,

$$Q_i(k) = \sum_{j \in V_i} \sum_{\ell=1}^{L} C(i,k,j,\ell)p_j(\ell) \tag{2}$$

at the first stage of hierarchy. Similarly for the second stage of hierarchy we obtain,

$$Q_i(k) = \sum_{\ell_1=1}^{L} \sum_{\ell_2=1}^{L} C(i,k,i_1,\ell_1,i_2,\ell_2)p_{i_1}(\ell_1)p_{i_2}(\ell_2) \tag{3}$$

with $q_i(k)$ defined by (1).

The global criterion that measures the consistency and ambiguity of the labeling over the set of units of template X is given by

$$c = \sum_{i=1}^{N} \vec{p}_i \cdot \vec{q}_i \tag{4}$$

let $\vec{v}$ be the vector of $R^P = R^L x \ldots x R^L$ $(P=NL)$ equal to $(\vec{p}_1, \vec{p}_2, \ldots, \vec{p}_N)$. Then (4) can be written as

$$c = \sum_{i=1}^{N} J_i(\vec{v}) \qquad (5)$$

where

$$J_i(\vec{v}) = \vec{p}_i \cdot \vec{q}_i$$

Now the total criterion of consistency and ambiguity for all the templates will be given by,

$$F(\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_M) = \sum_{m=1}^{M} \sum_{i=1}^{N_m} J_i(\vec{v}_m) \qquad (6)$$

where N_m is the number of segments and v_m is the P_m $(=N_m L)$ dimensional probability vector associated with the mth template X_m.

The condition that any segment of two different templates should not be assigned to the same segment of the object can be written as,

$$G(\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_M) = \sum_{i=1}^{M} \sum_{j=i+1}^{M} g(\vec{s}_i, \vec{s}_j) = 0 \qquad (7)$$

where $\vec{s}_\ell$ is obtained from $\vec{v}_\ell$ with the elements corresponding to the nil class set equal to zero for all the units of the template X_ℓ and $g(\vec{s}_i, \vec{s}_j)$ is the inner product of every vector $\vec{p}_i$ of $\vec{s}_i$ with every vector $\vec{p}_j$ of $\vec{s}_j$. What this condition essentially means is that if a unit i of a template X_m belongs to the class O_ℓ (where $\ell \neq L$), then sum of the inner product of the probability vector of this unit $\vec{p}_{im}$ with the probability vectors of all the units of all the templates should be zero. The nil class components have been excluded in obtaining $\vec{s}_\ell$ from $\vec{v}_\ell$, because one or more segments of different templates may belong to the nil class. Let us consider an example, when there are two templates, i.e., M=2. Also let the number of segments in template X_1 and X_2 be 3 and 4 respectively, then

$$G(\vec{v}_1,\vec{v}_2) = g(\vec{s}_1,\vec{s}_2) = 0 \tag{8}$$

$$\begin{aligned} g(\vec{s}_1,\vec{s}_2) &= (\sum_{i=1}^{3} \vec{p}_{i1}) \ (\sum_{i=1}^{4} \vec{p}_{i2}) \\ &= (\vec{p}_{11}+\vec{p}_{21}+\vec{p}_{31})\cdot(\vec{p}_{12}+\vec{p}_{22}+\vec{p}_{32}+\vec{p}_{42}) \\ &= \vec{p}_{11}\cdot(\vec{p}_{12}+\vec{p}_{22}+\vec{p}_{32}+\vec{p}_{42})+\vec{p}_{21}\cdot(\vec{p}_{12}+\vec{p}_{22}+\vec{p}_{32}+\vec{p}_{42})+ \\ &\quad \vec{p}_{31}(\vec{p}_{12}+\vec{p}_{22}+\vec{p}_{32}+\vec{p}_{42}) \end{aligned} \tag{9}$$

where

$$\vec{p}_{im} = [p_{im}(1),p_{im}(2),\ldots,p_{im}(L-1),0]^T$$

Now the segment matching problem can be stated as follows.

Problem Statement (A)

Given an initial labelings $\vec{v}_1^{(0)}$, $\vec{v}_2^{(0)},\ldots,\vec{v}_M^{(0)}$ for the set of M templates to belong to various segments of the object, find the labeling $\vec{u}_1$, $\vec{u}_2,\ldots,\vec{u}_M$ that correspond to the local maxima of criterion (6) which is closest to $\vec{v}_1^{(0)}$, $\vec{v}_2^{(0)},\ldots,\vec{v}_M^{(0)}$ subject to the following constraints.

(a) If $\vec{u}_m=(\vec{p}_{1m},\vec{p}_{2m},\ldots,\vec{p}_{Nm})$ then $\vec{p}_{\ell m}$ is a probability vector for $\ell=1,2,\ldots,N$ and $m=1,2,\ldots,M$. For a particular unit y of the template X_m, this means that the sum of the components of the vector $\vec{p}_{ym}$ be equal to unity and that each component be greater or equal or zero, i.e., if

$$\vec{p}_{ym} = [p_{ym}(1), p_{ym}(2), \ldots, p_{ym}(L)]^T$$

then

$$\sum_{\ell=1}^{L} p_{ym}(\ell) = 1$$

and

$$p_{ym}(\ell) \geq 0 \qquad \text{for } \ell=1,\ldots,L$$

(b) $G(\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_M)$ as defined by (7) be equal to zero.

Note that the criterion (6) is nonlinear. Constraint (a) involves linear equality and nonnegativity restriction and constraint (b) is nonlinear. In order to solve this optimization problem we use the penalty function concept [12-14] and modify the hierarchical shape matching gradient relaxation technique [10].

Coordinated Hierarchical Relaxation Technique Based On Projection Gradient Method and Penalty Function Approach

In order to solve the above problem using the penalty function approach, we define the penalized objective function [12-14] as,

$$\psi_C(\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_M) = F(\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_M) + \sum_{i=1}^{M} \sum_{j=i+1}^{M} d_{ij} \Phi_{ij}(g(\vec{s}_i, \vec{s}_j)) \quad (10)$$

where Φ_{ij} is a penalty function and $\{d_{ij}\}$ are penalty constants, also referred as the coordination factors. Since the constraint (b) given by (7) is an equality constraint, the penalty function is taken as the simple quadratic loss function given by

$$\Phi_{ij}(a) \triangleq -a^2 \tag{11}$$

Now the problem described in the earlier section becomes equivalent to that of maximizing (10) subject to the constraint (a). It can be solved using the gradient projection method applied to the linear constraints. We have used this method in the hierarchical shape matching algorithm described in [10]. We modify this algorithm with respect to the penalized objective function. Maximization of (10) subject to the constraint (a) is equivalent to maximizing

$$\begin{cases} \max\limits_{\vec{v}_1} F(\vec{v}_1)+S(\vec{v}_1,\ldots,\vec{v}_M) \\ \max\limits_{\vec{v}_2} F(\vec{v}_2)+S(\vec{v}_1,\ldots,\vec{v}_M) \\ \vdots \\ \max\limits_{\vec{v}_M} F(\vec{v}_M)+S(\vec{v}_1,\ldots,\vec{v}_M) \end{cases} \tag{12}$$

where $S(\vec{v}_1,\ldots,\vec{v}_M)$ corresponds to the 2nd term of (10). The algorithm has been implemented in the serial fashion on the computer, first we maximize with respect to $\vec{v}_1$, then with respect to $\vec{v}_2$ and so on. The main modification of the algorithm [10] with respect to this problem is the computation of the gradient. Earlier in [10] we maximized $F(\vec{v})$ with respect to $\vec{v}$, but now the contribution to the gradient also comes from the second term in (12), for example,

$$\text{new gradient with respect to } \vec{v}_1 = \text{old gradient with respect to } \vec{v}_1 + \frac{\partial}{\partial \vec{v}_1}[S(\vec{v}_1,\vec{v}_2,\ldots,\vec{v}_M)]$$

As an example let M=2, then the problem becomes to maximize

$$\psi_c(\vec{v}_1,\vec{v}_2) = F(\vec{v}_1,\vec{v}_2)-d(g(\vec{s}_1,\vec{s}_2)^2$$

subject to the constraints (a). d_{ii} is taken as d in the above equation. In order to solve this we consider

$$\left[\begin{array}{l} \max\limits_{\vec{v}_1} F(\vec{v}_1)-d[g(\vec{s}_1,\vec{s}_2)]^2 \\ \max\limits_{\vec{v}_2} F(\vec{v}_2)-d[g(\vec{s}_1,\vec{s}_2)]^2 \end{array}\right.$$

and use the method described in [10] with the modification for the gradient term. Similarly, when M=3, we solve the following,

$$\text{let } A = d_{12}[g(\vec{s}_1,\vec{s}_2)]^2+d_{13}[g(\vec{s}_1,\vec{s}_3)]^2+d_{23}[g(\vec{s}_2,\vec{s}_3)]^2$$

$$\left\{\begin{array}{l} \max\limits_{\vec{v}_1} F(\vec{v}_1)-A \\ \max\limits_{\vec{v}_2} F(\vec{v}_2)-A \\ \max\limits_{\vec{v}_3} F(\vec{v}_3)-A \end{array}\right.$$

In general to solve (12) by maximizing with respect to $\vec{v}_i$ the algorithm can be stated as follows:

<u>The Occlusion Algorithm</u>

1. Pick an initial estimate of $(\vec{v}_1^{(0)},\vec{v}_2^{(0)},\ldots,\vec{v}_M^{(0)})$. This is the initial assignment of probabilities to the units of the templates.

2. Pick the penalty constant $\{d_{ij}\}$ so that it provides a suitable balance between the associated first and second terms of (12).

3. Determine the maximum $\vec{v}_m^{(n+1)}$ $(m=1,2,\ldots,M)$ of the unconstrained penalized objective function (12) subject to the constraints (a) by using the present iterate $\vec{v}_m^{(n)}$ and projection gradient method [10].

4. Pick new penalty constants $\{d_{ij}\}$ in order to modify (or rebalance)

the magnitude of the penalty terms. The magnitude of the penalties should be increased to force a closer approach to the boundary; replace n+1 by n and return to 3.

Under the assumption of continuity of function F (6) and constraints (7) inherent in (10) the sequence of maxima $\{\vec{v}_m^{(n+1)}\}$ for m=1,...,M generated by the above algorithm approaches a constrained maxima of the problem defined in (A). The iteration is terminated after the convergence is considered as adequate. Details of these numerical strategies have been described in [10] and the rate of convergence and the possible ill-conditioning near the boundary have been discussed in [12-14]. However, in our case since we are seeking only local maximas, these problems do not occur. In the next section we present an example of the segment matching where two actual objects occlude each other to form an apparent object. Block diagram of the algorithm is shown in fig. 1.

Example

Figure 2 shows two actual objects which occlude each other to form an apparent object shown in fig. 3. Note that there is a change of scale for actual object X_1; it may result because of segmentation (different lighting conditions) or changes in the object itself even if the camera position remains fixed. In figs. 2 and 3 dotted points show the boundary of the objects and polygonal approximation has been obtained with a smoothing factor of 4. Table 1 shows the parameters used in the hierarchical gradient relaxation algorithm [10] and tables 2 and 3 show the results when only first stage is used. An inspection of figs. 2 and 3 gives the most logical assignment (see Table 4) for the units of X_1 and X_2. Note that for X_1 assignments are good, but for X_2, the assignment of unit 1 is wrong. It should be 19. Tables 5 and 6 show the results when the first stage with six iterations is followed by the second stage. Again, in this case assignments for X_1 are valid, whereas for X_2 assignment for unit 1 is wrong (Table 6). Although this assignment is correct at the second, third iterations of

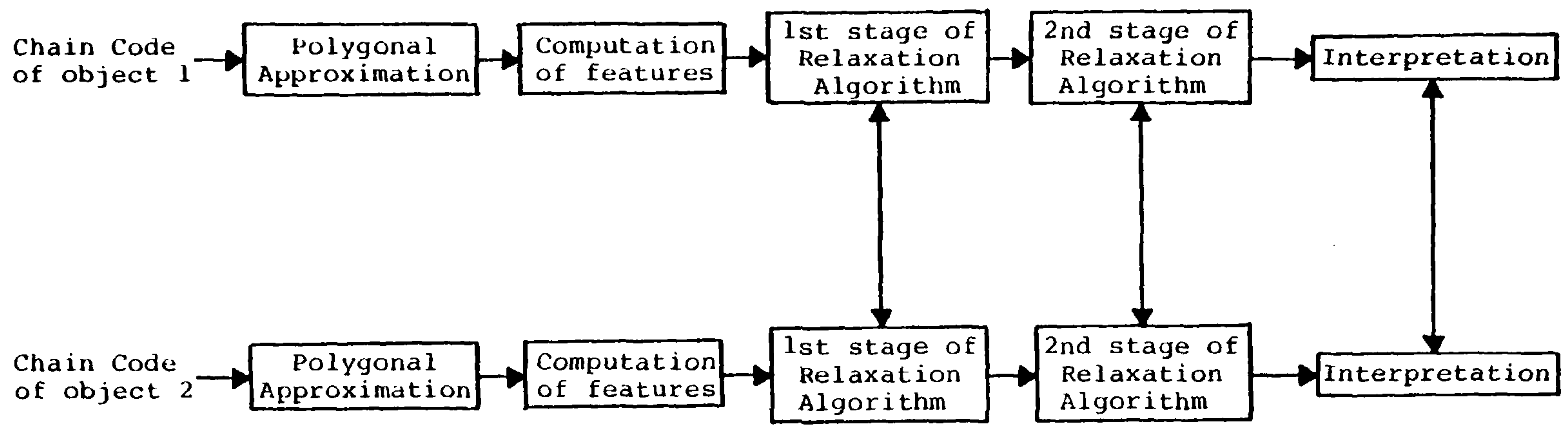
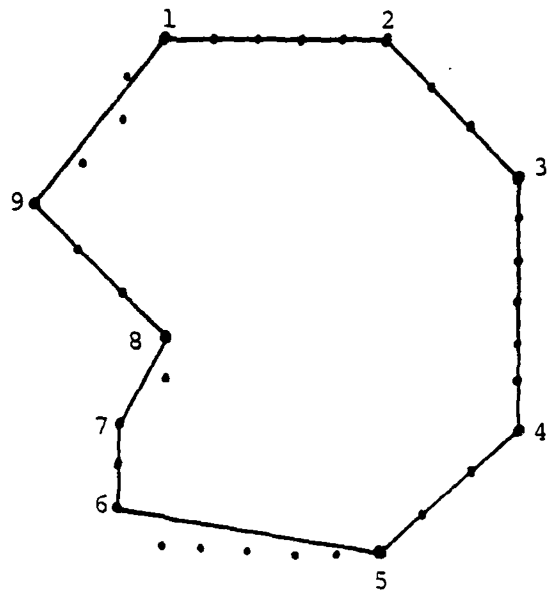
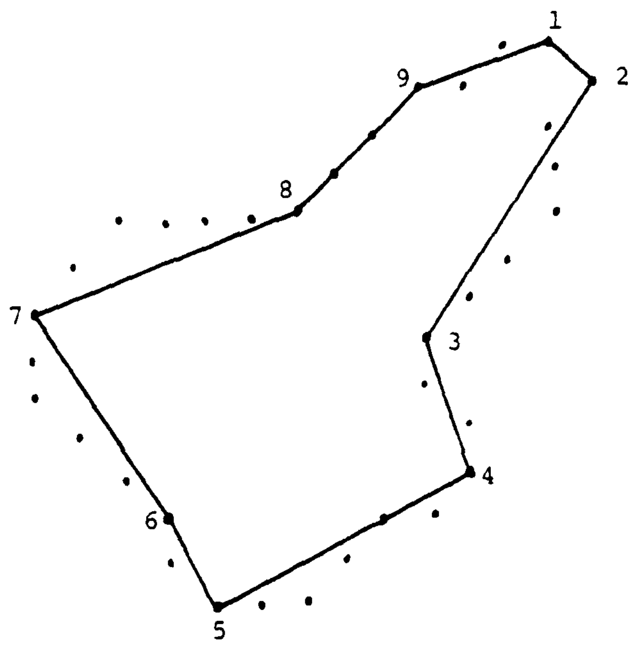


Figure 1. Block diagram of the occlusion algorithm for the shape description of 2 occluding objects using 2 stages of Coordinated Hierarchical Relaxation Technique based on Projection Gradient Method and Penalty function approach.



(a)



(b)

Figure 2. (a) First actual object X_1, perimeter = 34 points
(b) Second actual object X_2, perimter = 35 points
In each of the figures the number of vertices = 9 and dotted points show the boundary of objects. Polygonal approximation is obtained with a smoothing factor of 4.

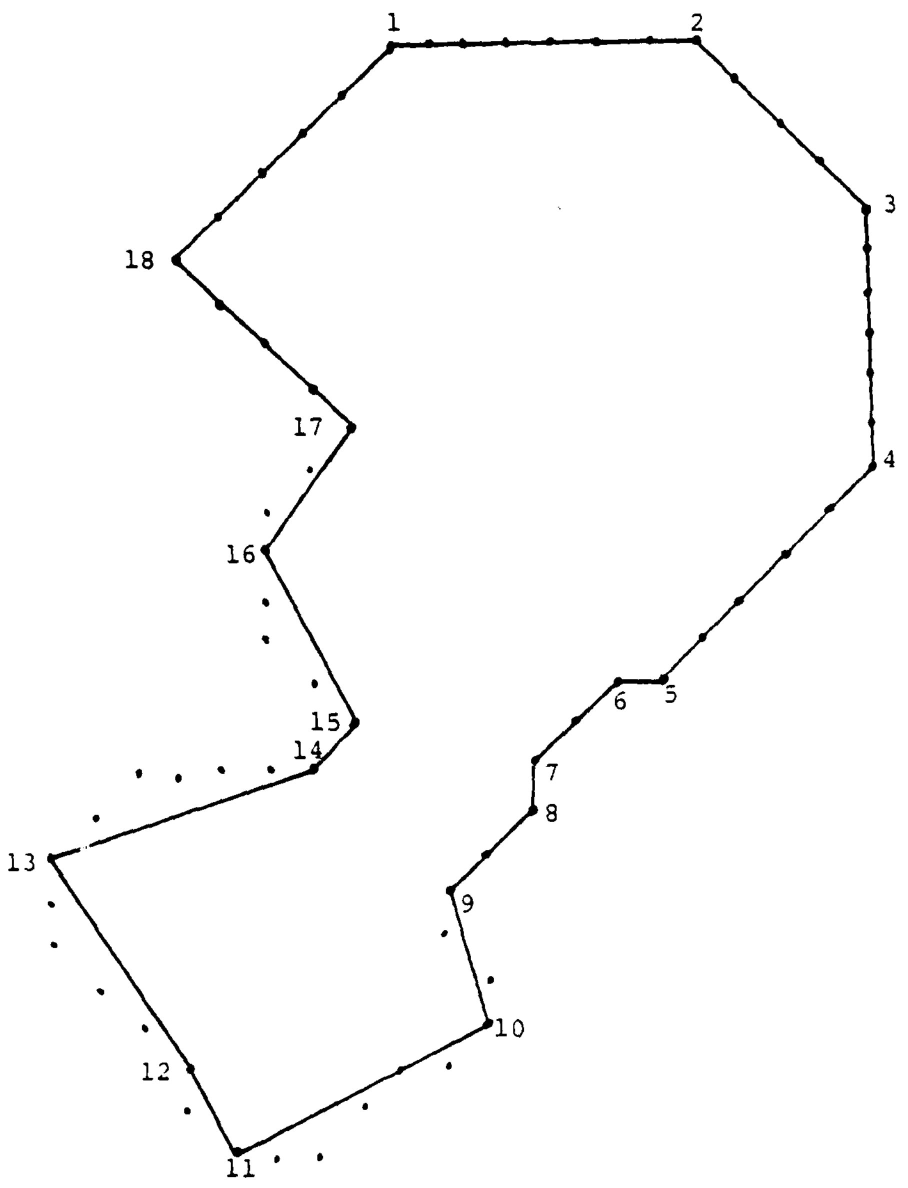


Figure 3. Apparent object formed as a result of occlusion of actual objects X_1 and X_2 shown in fig. 2. Perimeter = 67 points. Number of vertices = 18. Dotted points show the boundary of the object. Polygonal approximation is obtained with a smoothing factor of 4.

Table 1. Parameters used in the Hierarchical Relaxation Algorithm.

Smoothing factor in the Polygonal Approximation = 4

Weight of angle = 1

Weight of length = 2

Strength of angle = strength of length = 1

Initial probability assignment - slope and length of a segment is used

Nil class value for compatibilities and initial probability = 0.15

Compatibility computation - Average distance error method

Reduced compatibility and gradient computation -

Number of likely labels at the first and second stage = 1

Value of α in the iteration equation = 0.99

Upper threshold value for probabilities = 0.80

Lower threshold value for probabilities = 10^{-4}

Table 2. Assignment of units of actual *object* X_1 at various iterations of the first stage of Hierarchical Relaxation.

Unit	Assigned class at various iterations										
	0	1	4	7	10	11	12	13	14	15	16
1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3
4	6	6	6	6	6	6	6	6	6	6	4
5	5	19	19	19	19	19	19	19	19	19	19
6	19	19	19	19	19	19	15	15	15	15	15
7	16	16	16	16	16	16	16	16	16	16	16
8	17	17	17	17	17	17	17	17	17	17	17
9	16	16	16	16	18	18	18	18	18	18	18
Criterion		.945	1.140	1.178	1.242	1.333	1.392	1.352	1.413	1.429	1.466

Table 3. Assignment of units of actual object X_2 at various iterations of the first stage of Hierarchical Relaxation.

Unit	Assigned class at various iterations									
	0	1	3	5	7	9	11	13	15	16
1	2	2	2	2	2	2	2	2	2	2
2	4	4	4	4	4	4	4	4	19	19
3	9	9	9	9	9	9	9	9	9	9
4	10	10	10	10	10	10	10	10	10	10
5	11	11	11	11	11	11	11	11	11	11
6	12	12	12	12	12	12	12	12	12	12
7	13	13	13	13	13	13	13	13	13	13
8	18	14	14	14	14	14	14	14	14	14
9	13	13	13	13	13	19	19	19	19	19
Criterion		1.888	1.997	2.031	2.062	2.097	2.234	2.285	2.735	2.900

Table 4. Expected assignments of the units of actual objects X_1 and X_2.

Object X_1		Object X_2	
Unit	Class	Unit	Class
1	1	1	19
2	2	2	8 or 19
3	3	3	9
4	4	4	10
5	5 or 19	5	11
6	15 or 19	6	12
7	16	7	13
8	17	8	14 or 19
9	18	9	19

Table 5. Assignment of units of actual object X_1 at various iterations of the first and second stage of Hierarchical Relaxation.

Unit	Assigned class at various iterations										
	First stage				Second stage						
	0	1	3	6	1	3	4	6	7	8	10
1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3
4	6	6	6	6	6	6	6	6	4	4	4
5	5	19	19	19	19	19	19	19	19	19	19
6	19	19	19	19	19	19	19	19	19	19	19
7	16	16	16	16	16	16	16	16	16	16	16
8	17	17	17	17	17	17	17	17	17	17	17
9	16	16	16	16	16	16	18	18	18	18	18
Criterion		.945	1.121	1.163	.873	.905	.917	1.046	1.062	1.155	1.170

Table 6. Assignment of units of actual object X_2 at various iterations of the first and second stage of Hierarchical Relaxation.

Unit	Assigned class at various iterations									
	First stage				Second stage					
	0	1	3	6	1	2	3	6	8	10
1	2	2	2	2	2	19	19	2	2	2
2	4	4	4	4	4	4	19	19	19	19
3	9	9	9	9	9	9	9	9	9	9
4	10	10	10	10	10	10	10	10	10	10
5	11	11	11	11	11	11	11	11	11	11
6	12	12	12	12	12	12	12	12	12	12
7	13	13	13	13	13	13	13	13	13	13
8	18	14	14	14	14	14	14	14	14	14
9	13	13	13	13	13	13	19	19	19	19
Criterion		1.888	1.997	2.095	1.492	1.504	1.513	1.990	2.223	2.359

the second stage, but it is labeled as 2 at iteration 6 because unit 9 is labeled as 19 at iterations 3 which causes a change in the labeling of unit 1. One other fact that can be noted from careful examination of tables 2-6 is tha.. not only the number of iterations is little less when the second stage is followed by the first stage, but also it gives better labeling of units compared to the case when only first stage is used.

Finally, Tables 7 and 8 show the result of the occlusion algorithm. We have shown the values of the unconstrained objective function (first term of (10)) and penalty function term (second term of (10)) and the penalty constant at various iterations. Penalty constant is chosen such that it provides a balance between these two terms. For both objects X_1 and X_2 assignments are valid, however, the convergence rate is little slower than in tables 5 and 6.

For the unit 1 of X_2 the assignment does not change from 19, once it has been assigned (see table 6 and 8) and it has been found that the probability of this assignments increases with the iterations. Thus label 2 is uniquely assigned to the unit 2 of X_1. Also it is interesting to note the assignment of unit 2 of X_2 in tables 6 and 8.

Conclusion

In this paper we have investigated the problem of shape description of occluded objects based on segment matching. The matching has been done using an earlier shape matching algorithm [10] which has been modified to include the conditions for sucessful segment matching when objects occlude. The algorithm based on a hierarchical gradient projection method and penalty function concept is described and an example has been presented to illustrate the results. We hope that this framework provides a mathematical basis for the solution of occlusion problem.

Table 7. Assignment of units of actual object X_1 at various iterations of the Occlusion Algorithm.

Unit	Assigned class at various iterations										
	First stage				Second stage						
	0	1	4	6	1	2	3	4	6	8	10
1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3
4	6	6	6	6	6	6	6	6	6	4	4
5	5	19	19	19	19	19	19	19	19	19	19
6	19	19	19	19	19	19	19	19	19	19	19
7	16	16	16	16	16	16	16	16	16	16	16
8	17	17	17	17	17	17	17	17	17	17	17
9	16	16	16	16	16	16	16	16	16	18	18
Unconstrained objective function		.945	1.106	1.170	.883	.894	.914	.929	.953	.984	1.204
Penalty term of the objective function		.0789	.0921	.108	.126	.109	.0842	.0778	.163	.147	.128
Criterion		.886	1.015	1.061	.756	.784	.830	.851	.790	.837	1.075
Penalty constant		1	2	3	4	4	4	5	15	25	100

Table 8. Assignment of units of actual object X_2 at various iterations of the Occlusion Algorithm.

Unit	Assigned class at various iterations										
	First stage				Second stage						
	0	1	4	6	1	2	3	4	6	8	10
1	2	2	2	2	2	2	19	19	19	19	19
2	4	4	4	4	4	19	19	19	19	19	19
3	9	9	9	9	9	9	9	9	9	9	9
4	10	10	10	10	10	10	10	10	10	10	10
5	11	11	11	11	11	11	11	11	11	11	11
6	12	12	12	12	12	12	12	12	12	12	12
7	13	13	13	13	13	13	13	13	13	13	13
8	18	14	14	14	14	14	14	14	14	14	14
9	13	13	13	13	13	13	19	19	19	19	19
Unconstrained objective function		1.888	2.013	2.054	1.499	1.510	1.800	1.927	2.051	2.127	2.372
Penalty term of the objective function		.0789	.0921	.1082	.126	.109	.0842	.0778	.163	.147	.128
Criterion		1.809	1.921	1.946	1.372	1.400	1.718	1.849	1.888	1.979	2.243
Penalty constant		1	2	3	4	4	4	5	15	25	100

References

[1]. W.N. Martin and J.K. Aggarwal, "Dynamic Scene Analysis," Computer Graphics and Image Processing, Vol. 7, 1978, pp. 356-374.

[2]. H.H. Nagel, "Analysis Techniques for Image Sequences," Proc. 4th Int. Joint Conf. on Pattern Recognition, Nov. 1978, pp. 186-211.

[3]. M. Yachida, M. Asada and S. Tsuji, "Automatic Motion Analysis System of Moving Objects from the Records of Natural Processes," Proc. 4th Int. Joint Conf. on Pattern Recognition, Nov. 1978, pp. 726-730.

[4]. J.K. Aggarwal and R.O. Duda, "Computer Analysis of Moving Polygonal Images," IEEE Trans. Computers, Vol. C-24, Oct. 1975, pp. 966-976.

[5]. W.K. Chow and J.K. Aggarwal, "Computer Analysis of Planar Curvilinear Moving Images," IEEE Trans. Computers, Vol. C-26, Feb. 1977, pp. 179-185.

[6]. W.N. Martin and J.K. Aggarwal, "Computer Analysis of Dynamic Scenes Containing Curvilinear Figures," Pattern Recognition, Vol. 11, 1979, pp. 169-178.

[7]. H. Freeman, "Computer Processing of Line Drawing Images," Computing Surveys, Vol. 6, No. 1, March 1979, pp. 57-97.

[8]. R.O. Duda and P.E. Hart, "Pattern Classification and Scene Analysis," John Wiley and Sons, 1973.

[9]. J.W. Roach and J.K. Aggarwal, "Computer Tracking of Objects Moving in Space," IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 1, No. 2, April 1979, pp. 127-135.

[10]. B. Bhanu and O.D. Faugeras, "Shape Matching Using Hierarchical Gradient Relaxation Technique," Technical Report USCIPI 990,

Oct. 1980.

[11]. A. Rosenfeld and E. Johnston, "Angle Detection on Digital Curves," IEEE Trans. Computers, Sept. 1973, pp. 875-878.

[12]. C.N. Dorny, "A Vector Space Approach to Models and Optimization," John Wiley and Sons, 1975.

[13]. D.G. Luenberger, "Introduction to Linear and Nonlinear Programming," Addison-Wesley Publishing Co., 1973.

[14]. F.A. Lootsma, (ed.,), "Numerical Methods for Nonlinear Optimization," Chapter 23, Academic Press, New York, 1972.

[15]. B. Bhanu, "Computation of Features in the Analysis of Images of Moving Objects," Technical Report USCIPI 990, Oct. 1980.

2.9 Computation of Features in the Analysis of Images of Moving Objects

B. Bhanu

Introduction

Normally the input for the human vision system is a complex scene containing many objects with irregular shapes and the scene is changing due to the relative motion between the observer and the object. Different densities of receptors in the eye cause different actions by parts of the retina. Peripheral vision processes detect the motion and provide gross information about the scene. Detailed fine analysis is done by the attentive processes, wherever it is necessary [1]. Perception built over time, is the process of detecting the invariants of an object and finding distinctive differences. In other words, the perception of the scene is the result of integrating several input frames. Similar to a human vision system, a computer vision system capable of analyzing a sequence of frames should be able to extract information not only from each frame but from the sequence as much so that a description of the sequence can be obtained. Also the amount and type of processing should vary with the complexity at different places of the scene.

Movement is a form of change, and change is a significant perceptual attribute of our world. The basis of perceived change in human being lies entirely in the functioning of the nervous system. Change is a property which involves a comparison between what the object was and what the same object is now. Changes can be perceived in the location of an object, its structure, size, shape, color etc. [2]. Usually we can readily identify the nature of the perceived change. Similarly, a machine vision system for analyzing a sequence of images should detect significant changes in the features which distinguish the object from other similar objects. Continuing our work on the analysis of moving images [3], in this paper we describe a set of features which can be used in the analysis of such images and show the results on a test image.

Dynamic Image Understanding Model

The key problem in artificial intelligence is to automatically discover good distinguishing feature representations for objects based on general world knowledge. In order to address this problem and carry out the motion and shape analysis of objects in an adaptive manner we consider a new dynamic image understanding model reported earlier [3]. Its simplified block diagram is shown in Fig. 1. Model achieves the symbols, features and segmentation control by distributed feedback. Depending upon the matching of scene at a time with its prediction model, we do simple or complicated processing for motion as well as shape analysis. The prediction of later scenes should reduce processing subsequently. The features computed in a controlled fashion are used in a hierarchy of matching algorithms. Note that the segmentation varies with the dynamics of the scene. Now let us consider the meaning of the feedback conditions A, B, C, D, E shown in Fig. 1.

1) Want to compute more features for matching or use a different matching algorithm (A or B).

2) Want a better matching algorithm or a better segmentation (B or C).

3) When the matching of a scene with its prediction model is successful, then the prediction of the next scene will be used to carry out the segmentation (E).

4) More symbols/features are computed, but it is concluded that the incorporation of them in the matching algorithm will not lead to successful matching, then we want to go for better segmentation (D).

Work along the above lines is presently under investigation and will be reported in the future. In the next section we describe various features which should be useful in the analysis of images of moving objects.

Computation of Features

We want to use features for the recognition of objects and comparison of sequence of images to determine changes. Some of these features are similar to those used in image understanding by humans, while others are not such as moments. We classify the features into three categories:

1) Basic features such as area, perimeter, centroid, orientation etc.

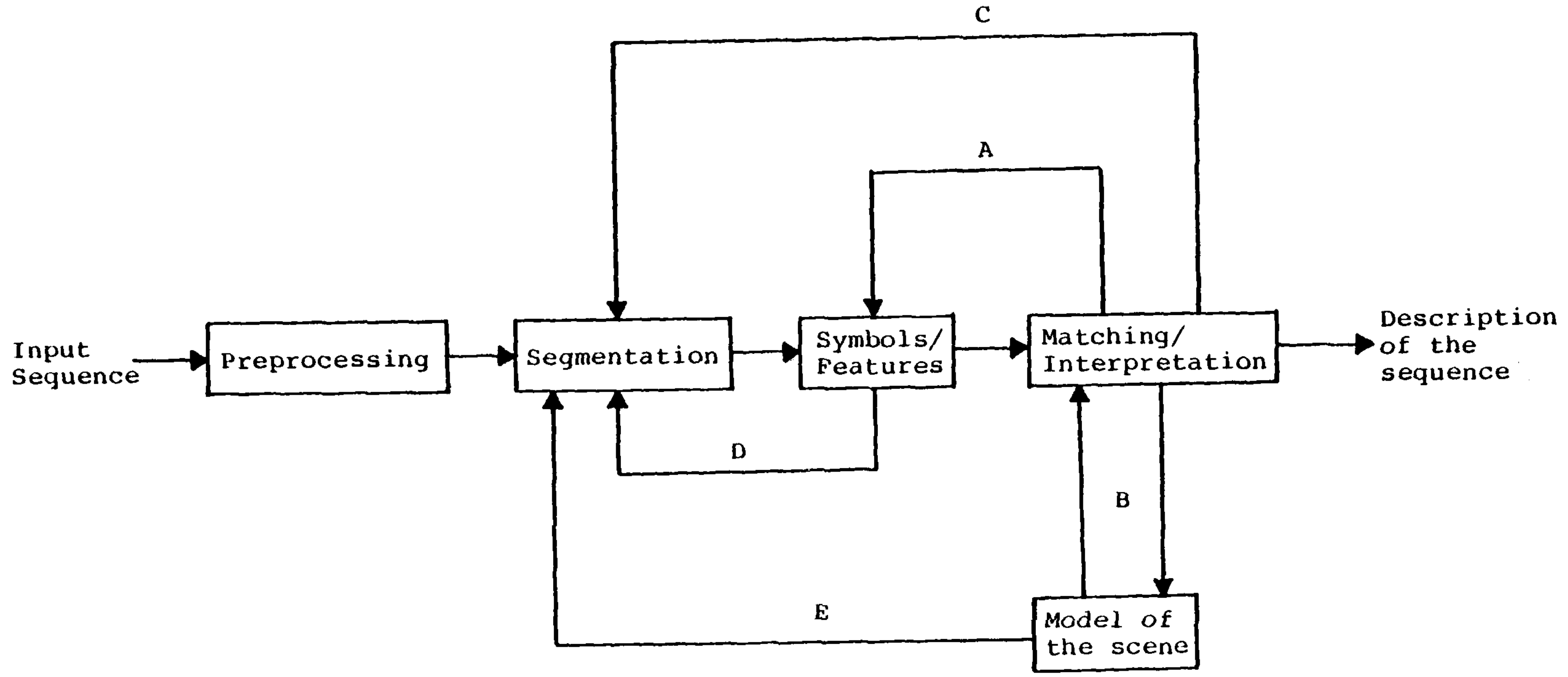


Fig. 1. System for the analysis of image sequences

2) Derived features are those which can be obtained from basic features such as the ratio of area/perimeter^2.

3) Contextual features are those which involve the relations with other objects such as relative position, size, neighbors, etc.

While these features are important in their own right, their rate of change should be useful in developing the prediction model. We are concerned here with the monochrome images and no textural features are used. However, if color and texture information is available, it should help in the analysis rather than making it more complicated. Results will be presented in the final section on a sequence of test images shown in Fig. 2. This figure consists of eight images of size 256x128 pixels and they are generated using the system described in [3].

Once the segmentation has been done it is assumed that the boundary of the objects are represented by chain code since it is a compact representation from storage view point [4]. However, to get the chain code for a segmented object (binary picture) may cause some problems because chain code requires the unique successor and predecessor of a boundary point. (A boundary in a digital plane is a collection of points where each pont is connected to two of its 8-neighbors, except for edge points where "Forks" exist). In order to overcome these problems we employ the following rules-

1. If all the 4-neighbors of a pixel are zero, then throw it away, i.e., we remove the isolated points.

2. If any of the 4-neighbors of a pixel are zero, then it is considered a boundary point [5, p. 339].

3. If any of the 4-neighbors of a boundary point obtained in step 2 are inside the boundary, then it will be the boundary point, otherwise

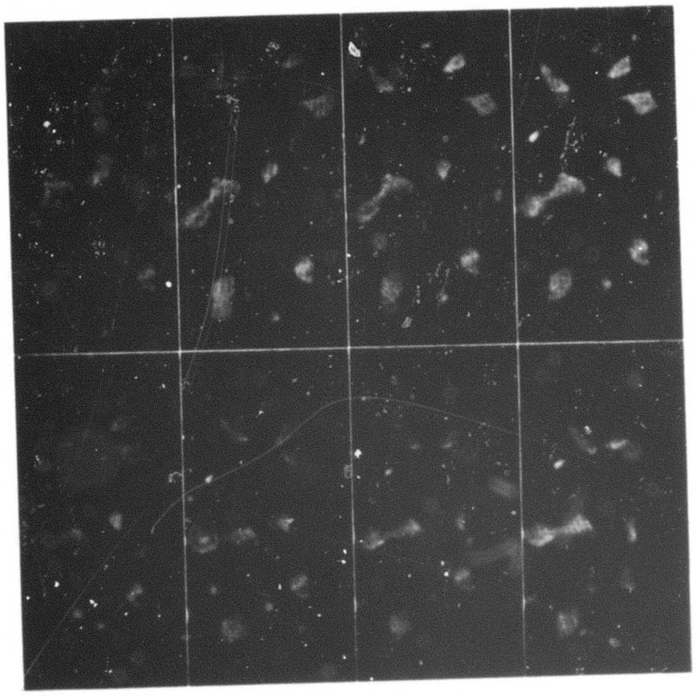


Fig. 2. A test input sequence of 8 images. The first row contains the first 4 images of the sequence and the second row contains the next 4 images. Size of each image of the sequence is 256x128.

we throw it away [6, p. 62]. This smoothing step eliminates the area of an object which is 2 pixels wide.

Even after the application of the above 3 rules we don't have unique successor and predecessor for every boundary point. Pathological cases do occur. In such cases we select one of the successors and proceed as usual by creating a stack. If no successor situation is encountered, then we start from the last situation where there were more than 1 successors and proceed. We keep repeating it until we get the chain code for the complete boundary of the object.

An object in the sequence is identified by the scene number and the object number in the scene. Object will be described by its starting point and chain code and it is noted whether the object is a boundary object (i.e., touches the boundary of the frame) or its is completely inside the frame. Note that from the chain code and starting point, we can uniquely find the coordinates of the boundary points, whenever necessary.

Computation of Basic and Derived Features

Size and Shape

The size of an object includes features such as width, length, maximum limits of extent (maximum and minimum X- and Y- values), area etc. For an object we define,

XMAX - Maximum X value
XMIN - Minimum X value
YMAX - Maximum Y value
YMIN - Minimum Y value

From the coordinates of the boundary XMAX, XMIN, YMAX, and YMIN can be easily obtained. Also X and Y center of the enclosing rectangle can be obtained as,

$$\text{X Center of Rectangle} = \frac{XMAX + XMIN}{2}$$

and

$$\text{Y Center of Rectangle} = \frac{YMAX + YMIN}{2}$$

Furthermore, width and length of the rectangle can be obtained as,

$$\text{width} = XMAX - XMIN$$

and

$$\text{length} = YMAX - YMIN$$

and

$$\text{Aspect ratio} = \text{width/length} = \frac{XMAX - XMIN}{YMAX - YMIN}$$

The area of an object is just the number of points that it covers. This is computed by counting the number of points inside the object. Area is also needed in the computation of some other basic features such as centroid, orientation etc.

<u>Shape</u>

There are almost limitless ways in which perceived shape may be classified, for example, circularity, angularity, elongation, symmetry, complexity and so on. However, we are concerned with those features which can be easily computed. Commonly used features are perimeter, the ratio area/perimeter^2, orientation, moment of inertia, length of the radius vector from centroid to the perimeter, a count of the number of corners of the shape etc. Now we shall elaborate on these features.

Perimeter: It is just the number of boundary points of an object. The ratio Area/Perimeter^2 is a measure of compactness of the object. It is a dimensionless quantity. It will be maximum for circles in a continuous world.

Orientation: The aspect ratio as described above suffers from the fact that it is dependent upon the orientation of the object. We would like to obtain the rectangle which just encloses the shape in arbitrary orientations such that the length of the rectangle is parallel to the principal axis. Direction of the principal axis is the direction of the major axis and the moment of inertia is minimum along this axis. Let θ be the angle between the principal axis and the horizontal axis. Let $\overline{X}$ and $\overline{Y}$ be the centroid of the object, then

$$\overline{X} = \frac{\sum_x \sum_y x f(x,y)}{\sum_x \sum_y f(x,y)} \qquad \overline{Y} = \frac{\sum_x \sum_y y f(x,y)}{\sum_x \sum_y f(x,y)}$$

$$m_{20} = \sum_x \sum_y (x-\overline{X})^2 f(x,y)$$

$$m_{02} = \sum_x \sum_y (y-\overline{Y})^2 f(x,y)$$

$$m_{11} = \sum_x \sum_y (x-\overline{X})(y-\overline{Y}) f(x,y)$$

where f(x,y) is the characteristic function of the object, which is 1 inside or on the object boundary and zero outside. Note that the denominator in the equations for $\overline{X},\overline{Y}$ is the area. The orientation θ is given by,

$$\tan 2\theta = \frac{2m_{11}}{m_{20}-m_{02}} \qquad (1)$$

It is to be noted that θ has an ambiguity of π radians. Because of this ambiguity sometimes it is useful to define the orientation nearest the angle of maximum radius vector. Now

Major axis = length of the orientation independent rectangle

$$= \max_{(x_1,y_1)} \{(x_1-\bar{X})\cos\theta+(y_1-\bar{Y})\sin\theta\}$$
$$+ \max_{(x_2,y_2)} \{(\bar{X}-x_2)\cos\theta+(\bar{Y}-y_2)\sin\theta\}$$

where (x_1,y_1) and (x_2,y_2) range over all the boundary points. [illegible] axis which is equal to the width of the orientation independent rectangle is obtained from (2) with θ replaced by $\theta+\pi/2$.

The moment of inertia of $f(x,y)$ about the line $y=x\tan\theta$ [illegible] centroid of $f(x,y)$ is taken as origin) is given by

$$m_\theta = \sum_x \sum_y (y\cos\theta-x\sin\theta)^2 f(x,y) \qquad (3)$$
$$= m_{20}\sin^2\theta+m_{02}\cos^2\theta-2m_{11}\sin\theta\cos\theta$$

(3) gives the minimum moment of inertia. Maximum moment of inertia is found from (3) when θ is replaced by $\theta+\pi/2$. The ratio of minimum to maximum moment of inertia is a measure of elongation.

<u>Radius vectors</u>: The length of the radius vector from centroid to the points on the perimeter describes the shape of an object. Important radius vector information includes the length of the maximum radius

vector and its orientation, length of the minimum radius vector and its orientation and length of the average radius vector. The derived features from these basic features include the angular difference between the maximum and minimum length radius vector, and a number of ratios of maximum, minimum and average length radius vectors.

Moments: Hu [7] has derived the seven moments which are invariant to rotation, translation and scale. Although these moments don't carry any physical intuition, but they are quite useful. For the sake of completeness, they are given below,

$$Q_1 = \eta_{20}+\eta_{02}$$

$$Q_2 = (\eta_{20}-\eta_{02})^2+4\eta_{11}^2$$

$$Q_3 = (\eta_{30}-3\eta_{12})^2+(3\eta_{21}-\eta_{03})^2$$

$$Q_4 = (\eta_{30}+\eta_{12})^2+(\eta_{21}+\eta_{03})^2$$

$$Q_5 = (\eta_{30}-3\eta_{12})(\eta_{30}+\eta_{12})[(\eta_{30}+\eta_{12})^2-3(\eta_{21}+\eta_{03})^2] \\ +(3\eta_{21}-\eta_{03})(\eta_{21}+\eta_{03})[3(\eta_{30}+\eta_{12})^2-(\eta_{21}+\eta_{03})^2]$$

$$Q_6 = (\eta_{20}-\eta_{02})[(\eta_{30}+\eta_{12})^2-(\eta_{21}+\eta_{03})^2]+4\eta_{11}(\eta_{30}+\eta_{12})(\eta_{21}+\eta_{03})$$

$$Q_7 = (3\eta_{21}-\eta_{03})(\eta_{30}+\eta_{12})[(\eta_{30}+\eta_{12})^2-3(\eta_{21}+\eta_{03})2] \\ +(3\eta_{12}-\eta_{30})(\eta_{21}+\eta_{03})[3(\eta_{30}+\eta_{12})^2-(\eta_{21}+\eta_{03})2]$$

where

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}, \qquad \gamma = \frac{p+q}{2}$$

and

$$\mu_{pq} = \sum_{x} \sum_{y} (x-\overline{X})^{p}(y-\overline{Y})^{q} f(x,y)$$

It is to be noted that compared with the other basic features, the computation of these moments is costlier.

<u>Ratios of Discrete Fourier Transform (DFT) Coefficients as Shape Features</u>

Various researchers [6] have investigated the use of Fourier transform itself or the features derived from the transform to obtain the features which are invariant to rotation, translation and scale. In this study we use some shape features derived by Granlund [8] in continuous case. Because of the changes from frame to frame it is not possible to use these features to obtain rotation (initial point does not remain the same from scene to scene because object is moving and moreover it is obtained after segmentation). We use 4 shape features, which are representative of the form of the contour.

$$D22 = \frac{a_3 a_{-1}}{a_1^2}$$

$$D12 = \frac{a_2^2 a_{-1}}{a_1^3}$$

$$D13 = \frac{a_2^3 a_{-2}}{a_1^4}$$

$$D44 = \frac{a_5 a_{-3}}{a_1^2}$$

a_i's in the above equations are Fourier coefficients. These features are complex in general. We have computed these features using Goertzel algorithm [9] since it is faster by a factor of 2 than FFT and moreover it does not require that the transform be computed at all the points. Here we need only 8 DFT coefficients. The boundary points are put in a complex array and the signal is padded with zeros. The transform size is taken to be 1024. We cannot take the perimeter as the transform size since the perimeter of an object varies from scene to scene either because the segmentation is not perfect or the shape of the object changes.

Contextual Features

In this study we have used five contextual features. One of them is principally a size feature. It is the area of an object relative to other objects (relation is greater). The other 4 features are neighbors of the object, relative position of an object in relation to its neighbors, centroidal distance between an object and its neighbors and the minimum distance between an object and its neighbors.

Neighbors

The neighbors of an object are found by finding all those objects which are completely or partially within an area of twice the width and length of the rectangle of this object when the rectangle is placed about the centroid of this object. This method of finding neighbors takes into consideration only the size of the object. A better way would be to consider the size as well as the velocity of the object in the determination of neighbors. Note that neighborhood relation is not symmetric.

Relative Position

For all the objects which are neighbors of an object, we find the relative position of these neighboring objects with the object. Relative position is given by 4 relations, above, below, to the right and to the left. An object R1 is said to be <u>above</u> object R2, if

(Top(R1) < Top(R2)) and
(Bottom(R1) < Centroid X(R2)) and
(Right(R1) MIN Right(R2)) $\geq$ (Left(R1) MAX Left(R2))

An object R1 is said to be <u>below</u> object R2, if

(Bottom(R1) > Bottom(R2)) and
(Top(R1) > Centroid X(R2)) and
((Right(R1) MIN Right(R2)) $\geq$ (Left(R1) MAX Left(R2))

An object R1 is said to be to the <u>left of</u> object R2, if

(Left(R1) < Left(R2)) and
(Right(R1) < Centroid Y(R2)) and
((Top(R1) MAX Top(R2)) $\geq$ (Bottom(R1) MIN Bottom(R2))

Similarly, an object R1 is said to be to the <u>right of</u> object R2, if

(Right(R1) > Right(R2)) and
(Left(R1) > Centroid Y(R2)) and
((Top(R1) MAX Top(R2)) $\geq$ (Bottom(R1) MIN Bottom(R2))

Although the neighborhood relation is not symmetric, yet we have taken the relative position relation as symmetric. For example (see Table 2) although object 5 has no neighbor, yet it is said to be above 7 because object 7 has object 5 as its neighbor and object 7 is below object 5. Also note that although an object may have a particular neighbor, but there may not exist any relative position relation between them.

Centroidal distance between an object and its neighbors - it is the distance between the centroid of this object and its neighbors.

Minimum distance between an object and its neighbors - it is the minimum distance between the boundary of this object and its neighbor. In other words it will be the minimum distance travelled by one of these objects when they will start touching each other.

An Example

For the test sequence shown in Fig. 2, computation of all the features is shown in Table 1 and 2 which show the basic and derived features for object 1 and contextual features for all the objects in the first frame. These tables are self explanatory.

Conclusion

In this paper we have discussed the computation of various features which should be useful in the analysis of image sequences.

Table 1. Basic and derived features for object 1 of frame 1 (see Fig. 2).

No. of elements in the chain code: 27
Starting coordinates: (16,79)
000000007556554445343431111

Basic Features

1. XMAX: 23
2. XMIN: 16
3. YMAX: 88
4. YMIN: 75
5. Perimeter: 27
6. Area: 72
7. X Centroid: 19
8. Y Centroid: 82
9. Orientation: 1.817 Rad.
10. Major axis: 13.33
11. Minor axis: 7.040
12. Max. Moment of Inertia: 766.7
13. Min. Moment of Inertia: 235.2
14. Max. Radius: 7.071
15. Min. Radius: 3.000
16. Avg. Radius: 4.444
17. Angle Max. Radius: 278.13 Degs.
18. Angle Min. Radius: 180.00 Degs.
19. Moment 1: 13.91
20. Moment 2: 54.50
21. Moment 3: 2.042
22. Moment 4: 2.945
23. Moment 5: 7.215
24. Moment 6: 17.95
25. Moment 7: -.33708
26. Mag D22: .9911
27. Phase D22: .0000
28. Mag D12: .9933
29. Phase D12: .0000
30. Mag D13: .9867
31. Phase D13: .0000
32. Mag D44: .9648
33. Phase D44: .0000

Derived Features

1. X Center of Rectangle: 20
2. Y Center of Rectangle: 82
3. Width of the Rectangle: 7
4. Length of the Rectangle: 13
5. Width/Length: .5384
6. Thinness,Area/Perimeter^2: .9876
7. Major axis/Minor axis: 1.894
8. Max. Moment of Inertia/Min. Moment of Inertia: 3.259
9. Max. Radius/Min. Radius: 2.357
10. Max. Radius/Avg. Radius: 1.590
11. Min. Radius/Avg. Radius: .6749
12. Angular difference (Max. and Min. Radius): 98.130 Degs.

Table 2. Contextual features for frame 1 (see Fig. 2).

Object	greater in Area to object	Neighbors of the object	Relative position of the object	Centroidal distance between object and its neighbors	Minimum distance between object and its neighbors
1	none	2	above 2	(1-2) 23.32	(1-2) 12.04
2	1,5,8,11	1,4	below 1 above 4	(2-1) 23.32 (2-4) 40.49	(2-1) 12.04 (2-4) 22.82
3	1,2,5,6,8,11	none	-----	-----	-----
4	1,2,3,5,6,8,9,11	2	below 2	(4-2) 40.49	(4-2) 22.82
5	1	none	above 7	-----	-----
6	1,2,5,8,11	7	right of 7	(6-7) 48.76	(6-7) 8.00
7	1,2,3,4,5,6,8,9,10,11	5,6,9,10	below 5 left of 6 above 10	(7-5) 50.28 (7-6) 48.76 (7-9) 94.62 (7-10) 67.26	(7-5) 25.94 (7-6) 8.00 (7-9) 69.29 (7-10) 32.89
8	1	none	-----	-----	-----
9	1,2,3,5,6,8,11	11	-----	(9-11) 52.49	(9-11) 37.48
10	1,2,3,4,5,6,8,9,11	11	below 7	(10-11) 46.40	(10-11) 28.42
11	1,5,8	none	-----	-----	-----

The rate of change of feature values should be very helpful in generating more accurate and precise prediction model. The computation of these features is conditional as shown in Fig. 1. The work along these lines is presently under investigation and will be reported in the future.

References

[1]. J.J. Gibson, "The Perception of the Visual World," Houghton Mifflin, Boston, 1950.

[2]. K.E. Price, "Change Detection and Analysis in Multi-Spectral images," Ph.D. Thesis, 1976, Dept. of Comput. Sci., Carnegie Mellon University.

[3]. B. Bhanu and O.D. Faugeras, "Computer Analysis of Moving Images," USCIPI Report 960, March 1980, pp. 116-128.

[4]. H. Freeman, "Computer Processing of Line Drawing Images," Computing Surveys, Vol. 6, No. 1, March 1974, pp. 57-97.

[5]. A. Rosenfeld and A.C. Kak, "Digital Picture Processing," Academic Press, New York, 1976.

[6]. T. Pavlidis, "Structural Pattern Recognition," Springer-Verlag, 1977.

[7]. M.K. Hu, "Visual Pattern Recognition by Moment Invariants," IEEE Trans. Information Theory, Vol. IT-8, Feb. 1962, pp. 179-187.

[8]. G.H. Granlund, "Fourier Preprocessing for Hand Print Character Recognition," IEEE Trans. Computers, Vol. C-21, Feb. 1972, pp. 195-201.

[9]. G. Goertzel, "An Algorithm for the Evaluation of Finite

Trigonometric Series," Amer. Math. Monthly, Vol. 65, Jan. 1958, pp. 34-35.

2.10 Region Descriptions Using Range Data

A. Huertas, S. Inokuchi and R. Nevatia

Introduction

Scene analysis problems including segmentation and shape analysis are simplified when range data , i.e. the distances of observed points of a scene from the viewer, is available. An edge based approach has been described in [1] by which fairly complete region boundaries can be detected in range pictures. Boundary segments can then be traced in order to find closed regions and some region properties can be inferred by observing the orientation of the segments along a region. Descriptions on how regions occlude other regions can also be obtained.

In this report we describe a boundary tracing program which traces region boundaries that have been detected using the method described in [1]. The region descriptions provided by the program include a classification of the regions found as complete parts, subparts, complete holes and spaces. Occlusion information is obtained by examining the classification of the regions and the available range data.

Region Descriptions

The region description process proceeds in two steps.

a. Region boundary tracing

Finding boundaries of closed regions is relatively simple, if complete boundary segments are found. By maintaining a list of the segments with two locations for each segment, suitable segments can be selected to start the trace of a region. Once a segment is selected the boundary can be followed by selecting the appropriate sequence of "next segments". Each of the two positions in the list are used to mark a selected segment as "visited" according to the direction followed.

The process proceeds in a counterclockwise manner. The "next segment" function (NSF) selects among the set of segments with one end point lying within a one pixel neighborhood of the end point of the last traced segment (current segment), the one that has the smallest angle measured clockwise at the end point of the current segment if the segments intersect. Otherwise the angle is measured at the intersection of the extension of the current segment with the segment being considered for selection. Figure 1 shows several examples of how the angles are measured. If the selected segment is located within a certain distance of the frame of the picture, the frame is followed in the right, up, left or down direction according to the section of the frame encountered (bottom, right side, top or left side respectively). The first segment encountered with one end point lying within a certain distance of the frame is selected and the process continues.

Four aspects need to be considered when selecting the next segment in the tracing process.

1) Figure 1a shows that one pixel long segments might be skipped if they are collinear with the current segment. Segment S_1 is the current segment, S_2 is a one pixel long segment, and both S_2 and S_3 have an end point within one pixel of the end point of S_1. Since

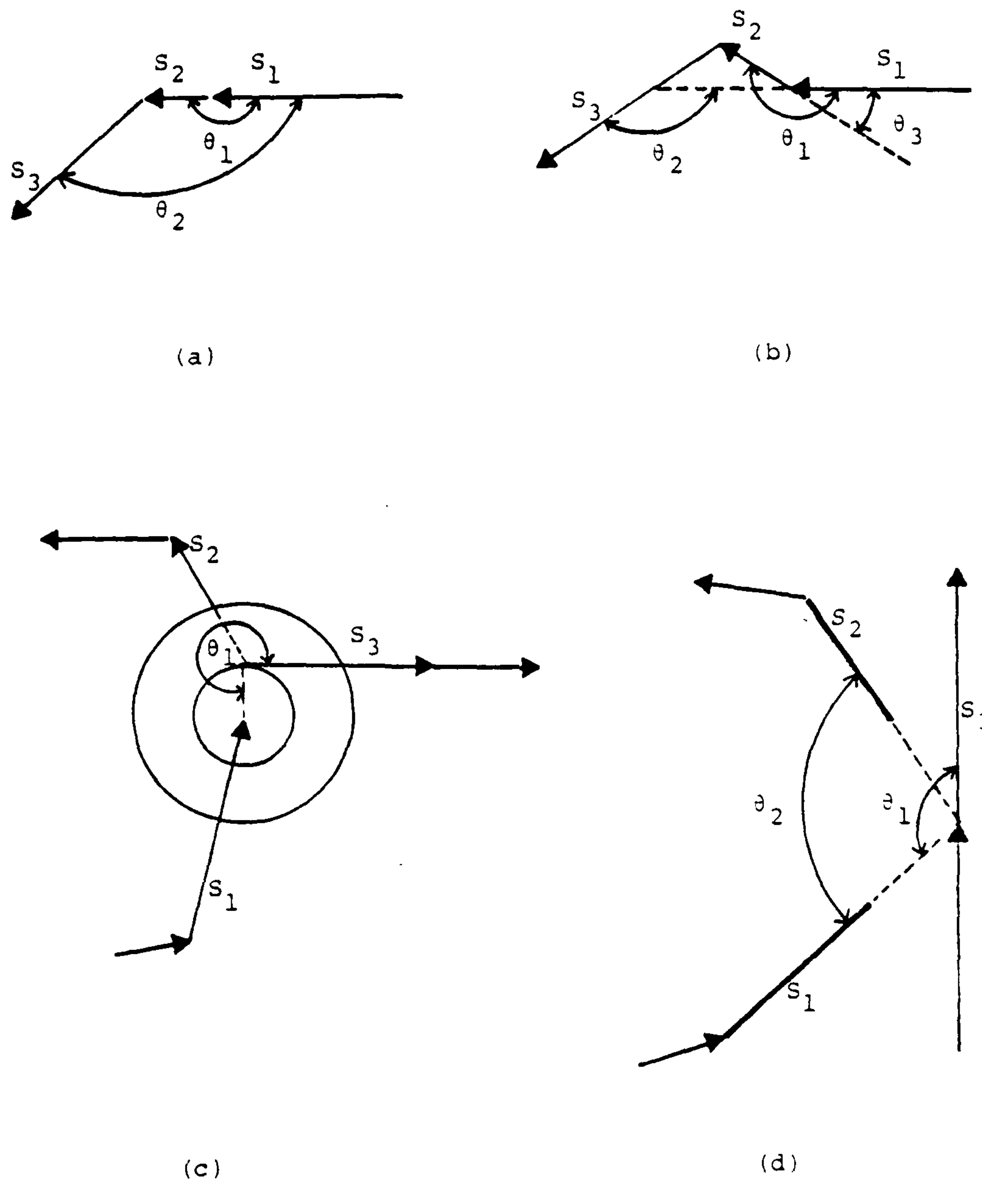

s_2
s_1
θ_1
s_3
θ_2
(a)
s_2
s_1
s_3
θ_2
θ_1
θ_3
(b)
s_2
s_3
θ_1
s_1
(c)
s_2
s_3
θ_2
θ_1
s_1
(d)

Figure 1.

$\theta_2 < {}_1$, S_3 is selected and S_2 is skipped. The NSF knows if a segment is skipped and marks it as visited to prevent its being selected again when starting the trace of a new region. Figure 1b shows that if a one pixel long segment is not collinear with the current segment, the direction of the trace might be reversed. S_1 is the current segment, S_2 is a one pixel long segment with both ends lying within one pixel from S_1, and S_3 has one end point within one pixel from S_1. The angle θ_3 between S_1 and S_2 measured at the head of S_1 taking S_2 in the direction opposite to the orientation of S_1 is usually very small. This clue is used by the NSF if a change in direction occurs to determine the selection or rejection of a one pixel long segment as a suitable next segment.

2) Regions that can be described as parts or subparts tend to be convex shaped. Figure 1c shows that the process can be side tracked if the angle θ_1 between the current segment S_1 and S_3 is larger than 180 degrees. To avoid this, the NSF places a window centered at the end of the current segment S_1. This window can grow up to three pixels in diameter in an attempt to locate a segment which preserves convexity.

3) If N is the size of the masks used during the boundary detection process, short segments lying within (N-1)/2 pixels of the frame of the picture are considered unreliable due to the incomplete data resulting from the convolution of the picture data with the masks. These segments are eliminated in a preprocessing step.

4) Non-oriented segments generated by the radial line detection process described in [1] fall one pixel short of the location where they would meet existing segments. These segments are shown in figures 1d and 3 as thick lines with no orientation. Figure 1d shows the current segment S_1, the segment S_3 with one end point within one pixel from S_1, and S_2 with one end point located two pixels from S_1. Notice that although segment S_3 meets all the criteria for a suitable next segment, the desired next segment is S_2. By extending the loose

end of the non-oriented segments by one pixel in the preprocessing step, the desired next segment can be easily selected.

When the process is unable to select a suitable next segment given the above considerations, the current segment is extended by up to a fixed number of pixels. At each step a suitable next segment is sought for. The trace of the current region is terminated if no suitable segment is found and the region being traced is classified as an incomplete part.

Two criteria determine a closed region. The starting point of a trace is within one pixel of the ending point and/or the sum of the exterior angles is approximately equal to 360 degrees in absolute value.

b. Region descriptions

Some region properties can be inferred by observing the orientation of the segments along the boundary of a region as shown in figure 2. Basically the parts and spaces can be differentiated by whether the segments in the boundary are oriented in the same direction as the trace or in the opposite direction. Related line labeling analysis can be found in [2].

Occlusion is indicated by not all segments pointing in the same direction, compared to the direction of the region tracing. By generating a set of triples (A,B,C) indicating that region A and region B share a common segment C, the following can be asserted:

- If the B component in a triple occurs only once among the set of all triples then region B occludes region A and region A cannot be merged with any other region to form a larger region. In our example shown in figure 3 the triple $(5,1,S_i)$ corresponds to region 5 (A component) being occluded by region 1 (B component). No other triple contains region 1 as its B component and therefore region 5 cannot be

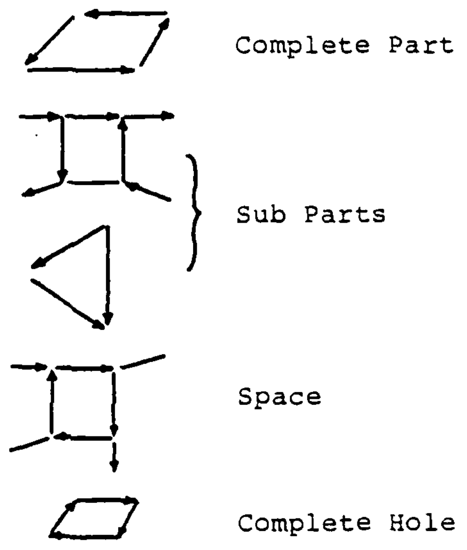
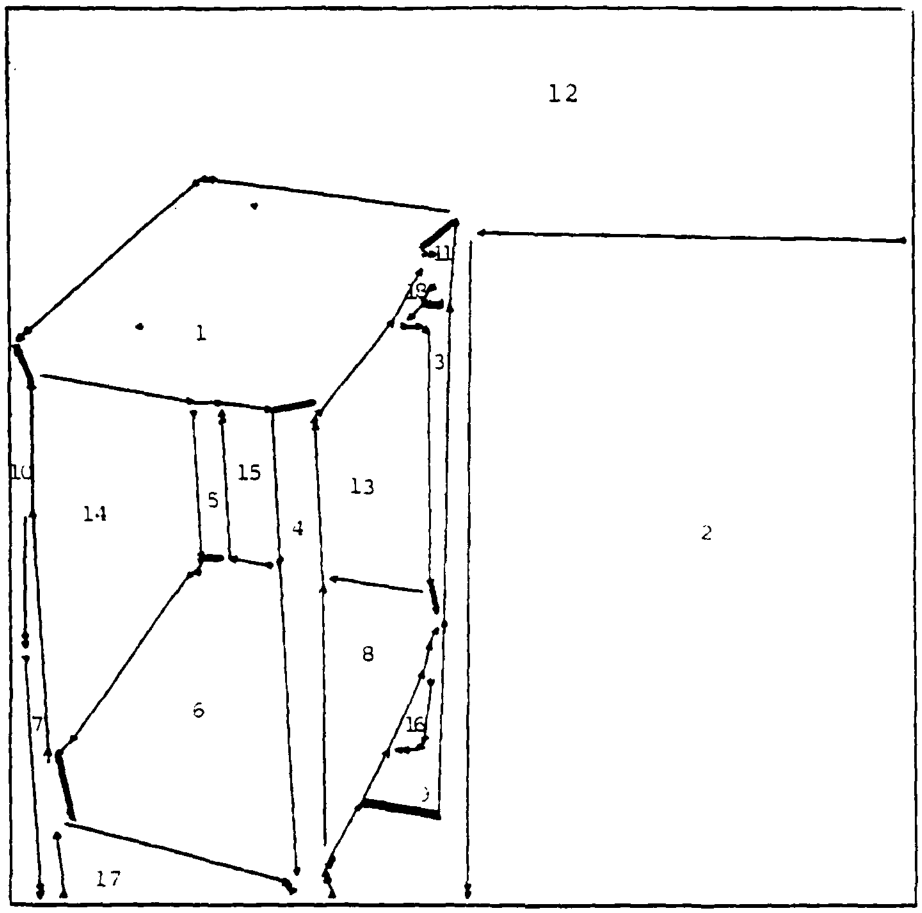


Figure 2. Types of regions.



CLASSIFICATION OF REGIONS

Region 1:	complete part	Region 10:	subpart
Region 2:	complete part	Region 11:	complete part
Region 3:	complete part	Region 12:	space
Region 4:	complete part	Region 13:	space
Region 5:	subpart	Region 14:	space
Region 6:	subpart	Region 15:	space
Region 7:	complete part	Region 16:	space
Region 8:	subpart	Region 17:	space
Region 9:	subpart	Region 18:	incomplete part

OCCLUSION

Parts occluding one part:
Subpart 5 is occluded by complete part 1
Subpart 6 is occluded by complete part 4
Subpart 8 is occluded by complete part 4
Subpart 9 is occluded by complete part 8
Subpart 10 is occluded by compete part 7

Parts occluding two parts:
Subparts 6, 8 are one part occluded by complete part 4

Figure 3.

merged with any other region.

- If the A component in a triple occurs only once among the set of all triples then only the region indicated by the B component occludes it. In our example the A component in the triple $(9,8,S_j)$ corresponds to region 9 and occurs only once. Therefore only region 8 occludes it.

- If two or more triples have the same B component, they can be merged to form a larger region provided they lie on the same plane. By finding some of the interior points in the occluded regions, the coefficients of the equations of their plane surfaces can be estimated. If the two planes are within certain tolerances of each other, then the occluded regions can be merged to form a larger region. In our example the triples $(6,4,S_k)$ and $(8,4,S_\ell)$ have the same B component. By estimating and comparing the coefficients of the planes on which regions 6 and 8 lie, it is determined that they can be merged to form a larger region.

- If two or more triples have the same A component then two or more regions occlude the region indicated by the A component. The occluding regions (B components) can be merged to form a larger occluding region if they lie in the same plane and have at least one segment in common. No instance of this case occurs in our example.

Figure 3 shows the regions in our example and the classification made by the program and the description of how regions occlude other regions.

References

[1]. S. Inokuchi and R. Nevatia, "Feature Extraction in Range Data," Image Processing Institute, USCIPI Technical Report 960, March 1980.

[2]. K. Sugihara, "Range Data Analysis Guided by a Junction

Dictionary," Artificial Intelligence Journal, Vol. 12, 1979, pp. 41-69.

3. HARDWARE IMPLEMENTATION OF IU ALGORITHMS

ADVANCED IMAGE UNDERSTANDING USING LSI AND VLSI

S.D. Fouse, V.S. Wong, and G.R. Nudd

Hughes Research Laboratories
Malibu, California 90265

Abstract

We describe here the work undertaken at the Hughes Research Laboratories, Malibu, in support of the DARPA Image Understanding (IU) program. This report covers the period from May 1980 through September 1980. The principal aim of our work during the present phase of the contract is to investigate the applicability and potential benefit of very large scale integrateion (VLSI) to IU systems. Our work towards this goal includes detailed logic design for two intermediate-level IU systems, a line finder and a texture classification system, and identification of a highly modular programmable digital processing element, which is currently under construction. In addition to the major emphasis of the program, we are directing our work so that the hardware designed will be fully compatible with existing commercial hardware such as the DEC mainframes. The details of the work that has been completed to date as well as our goals for the program are described below.

I. INTRODUCTION

The main emphasis of this program is the investigation of the impact and potential benefit of very large scale integration (VLSI) and high-density IC technologies for image understanding. The task is somewhat wider than our previous work for the Image Understanding Program, where we successfully developed special-purpose high-speed primitives for the low-level processing operations.

The progress that is currently being made in silicon technology and the development of more sophisticated algorithms for image understanding provide the basis for a major advance in processing capability. The work described here is being undertaken at Hughes Research Laboratories (HRL), where we are actively involved in the development of image analysis and understanding software for applications such as terminal homing, image bandwidth compression, and scene matching. We also have numerous active research programs in micro-electronic technology. The people involved in this program were also involved in the VHSIC-0 program. In addition at HRL we have eleven VHSIC-3 technology development programs. We are therefore aware of the major developments in these and other important programs, and, where appropriate, we have coordinated our efforts.

Our major emphasis has been to review numerous complex image-understanding algorithms and devise VLSI concepts for them. This includes understanding the processing, data flow, timing, and storage requirements. We have also performed a detailed partitioning of potential VLSI chips based on total gate count, silicon area, communication requirements, and power considerations. From this work, three potential systems have emerged: a line finder, texture analyzer, and segmenter. The details of this are included in this report together with the necessary gate estimate and parts count for VLSI chips (Tables 1 and 2).

In addition to this, we have, through our detailed analysis of the VLSI processor requirements and configuration, identified a highly modular programmable digital processing element RADIUS (residue-based arithmetic image-understanding system). These concepts are now being developed in state-of-the-art n-MOS technology, and we anticipate that the first demonstration will be available in the early part of 1981. Since RADIUS is based on residue arithmetic, it can perform a wide variety of processing operations (including variance and moment calculation) and all convolutions and local area operations with very high circuit function density. The modularity of our approach allows rapid and easy design in VLSI and provides programmability and portability.

In addition a major emphasis of all our work at this point is to ensure that the hardware concepts and changes will be fully integratable with existing commercial hardware, such as the DEC mainframes. To this end, we are developing the necessary interfaces to the RADIUS processor so that it can be used as an attached processor to the DEC series, communicating through the DEC-UNIBUS. We are obtaining a PDP11-34 to use as the test vehicle for our approach. We anticipate that this machine will be available in mid-1981, after which the successful development of our hardware and interface will provide a means for integrating the system into the DARPA IU test-bed as well as other systems.

Our work on more complex IU algorithms in addition to the three cited above will continue to enable us to identify special elements in the IU chain that can appropriately exploit the VLSI and VHSIC developments. This work, to be reported on in the next semi-annual, will include concept developments, partitioning, simulation, chip count, and where appropriate, layout details.

II. METHOD

Our objective for this project is to provide some general results that will allow the image understanding community to take full advantage of VLSI technology as it becomes available. We expect our results to include definitions for several special-purpose chips that will have wide applicability to IU systems. The question to be answered then is what types of functions should be cast into a VLSI chip. The goals of this program are analogous to the on-going VHSIC efforts where they are searching for commonality across a broad range of DOD systems. Here we are trying to take a longer range view specifically for image understanding.

The project has been coordinated with the Hughes VHSIC program and shares many goals with it. The major difference is that we would like to see commonality across IU systems. We are tracking the VHSIC program, and, where appropriate, we will be able to take advantage of the work being done there that complements our own effort.

The area of this study is necessarily somewhat broad. It differs considerably from our previous work, where we developed special-purpose high-speed primitives for IU. Our aim this period has been to stand back and take a broad a vew of IU requirements and to attack the question of what special-purpose VLSI functions would be appropriate for IU systems that will not be developed under present or foreseeable VHSIC programs. Our approach is to

- Select three representative systems to study and characterize
- Perform a commonality study across the three systems
- Do logic designs for each system
- Partition the designs onto VLSI chips
- Test and refine designs using simulation techniques
- Identify relevant system parameters
- Generalize to additional systems and refine function partitioning.

So far, we have selected three systems and performed a preliminary commonality study, identifying one subsystem as a likely candidate for a VLSI chip. This is a local area processor which will perform sliding window arithmetic operations including convolution, variance, and moment calculations. We have designed a processor based on the residue arithmetic technique that should provide significantly better performance for this type of operation than a binary processor. In addition, we have performed logic designs on two of the three systems selected: the line finder and the texture classification systems. What remains to be done is to design the third system, the segmenter, refine our designs using results from simulations, and then extend the results to additional IU systems.

III. RESIDUE-BASED ARITHMETIC IMAGE UNDERSTANDING SYSTEM (RADIUS)

Almost all of the systems we have looked at require some sort of local-area processing where the output pixel is a function of the input pixel and its M nearest neighbors, where M is usually 8, 24, 48, etc. The function takes numerous forms but typically is either arithmetic or logical or a combination of both. If the function is arithmetic and does not require division or absolute value, then residue arithmetic techniques can be used. Examples where these conditions are met can be found in two of the systems we are studying: the line finder and the texture classification systems. The line finder detects edges by convolving the image with six 5 x 5 masks, each mask responding to a different direction. Since the convolution operation requires multiplications and additions, this is easily done using residue techniques. Similarly, the texture classification system convolves the input image with several 5 x 5 masks.

An arithmetic local area processor seems to be a natural choice for a subsystem that can take advantage of VLSI technology. This is because of the complexity of the logic for doing multiples. Using current technology, a high-speed multiplier requires an entire chip (such as the TRW 10-MHz multiplier). One approach being considered for real-time hardware is to compromise on the coefficients and thereby reduce the complexity of the hardware. The residue techniques can offer significant advantages without compromise, since multiplication can be performed using look-up tables; these tables will be small because the bases used will be small. As an illustration of this, Figure 1 shows the components involved in RADIUS. The input data are initially encoded into its equivalent representation in each base. This is most easily done using a ROM lookup table with an address space equal to the input dynamic range and a bit depth equal to the number of bits required to represent the Base-1. The actual computation is then performed by the local area processors, one processor per base used. The processor output word size is identical to the input word size with no loss in accuracy. This is due to the modular nature of residue arithmetic. The data out of the convolvers then goes into a residue-to-binary decoder. This block can

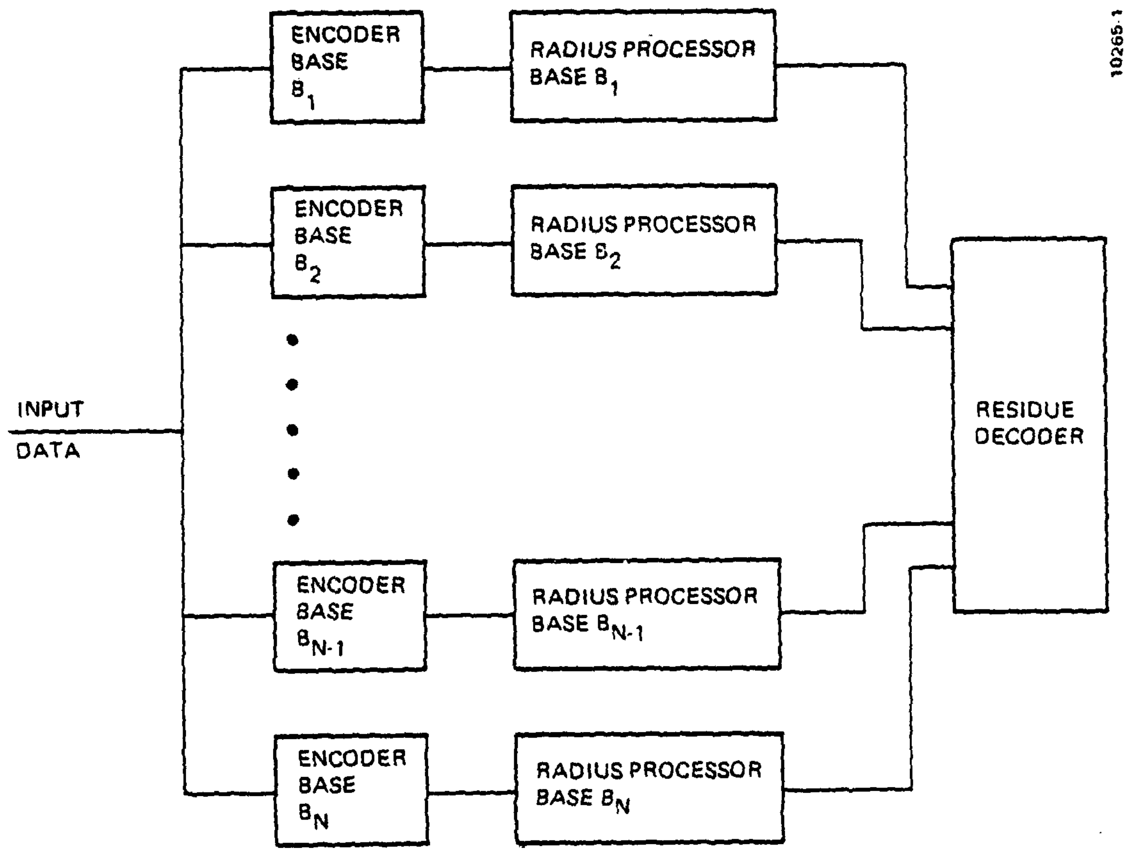

ENCODER BASE B_1
RADIUS PROCESSOR BASE B_1
ENCODER BASE B_2
RADIUS PROCESSOR BASE B_2
INPUT DATA
RESIDUE DECODER
ENCODER BASE B_{N-1}
RADIUS PROCESSOR BASE B_{N-1}
ENCODER BASE B_N
RADIUS PROCESSOR BASE B_N
10265-1

Figure 1. A residue-based processor.

be composed of a very large lookup table or a system composed of ROMs and adders. The exclusive use of look-up tables is well suited to VLSI since these are highly regular structures and hence can be made very dense and inexpensive.

Many arithmetic operations can be performed with table look-ups, but without the residue concept, this approach is typically not feasible, as the tables required become overwhelmingly large. Since both the line-finder system and the texture system utilize 5 x 5 convolutions, the processor system should be capable of being used to perform 5 x 5 convolutions with programmable weights. The dynamic range capabilities of the system should allow for 6 bit input and 6 bit kernel weights for a total output dynamic range of 17 bits. In addition, the system should be able to operate at a 10-MHz data rate. The dynamic range of a residue computation system is determined by the product of all the bases. Using 31, 29, 23, and 19 as bases will give us a dynamic range of 392,863, which exceeds 2^{17}. The bases will have other benefits since they are al prime numbers.

Figure 2 shows a block diagram for a 5 x 5 local area processor for a single base. There are five data inputs, one for each line of the 5 x 5 area. Each input data is put into a register which is the first element of a five-element shift register. When new data is transferred in, the previous data is transferred to the next register, and so on. There are five inputs and thus five shift registers with five elements each for a total of 25 registers. The contents of each of the 25 registers are used to address 32 element by 5 bit RAMs. These RAMs are lookup tables for the multiplication of the input data and the kernel weight. The outputs of the 25 RAMs are then added by a tree of 24 residue (or modular) adders. A residue adder calculates (A + B) mod base.

There are several points to notice about this design. First, only 5 bits come out of the multiply or add. This is because these operations are cyclical, and the output values are in the same range as each of the input values. The second point is that the only part of the processor that is dependent on the base is the residue adder. Since the multiplier must be programmed for the weights anyway, the choice of

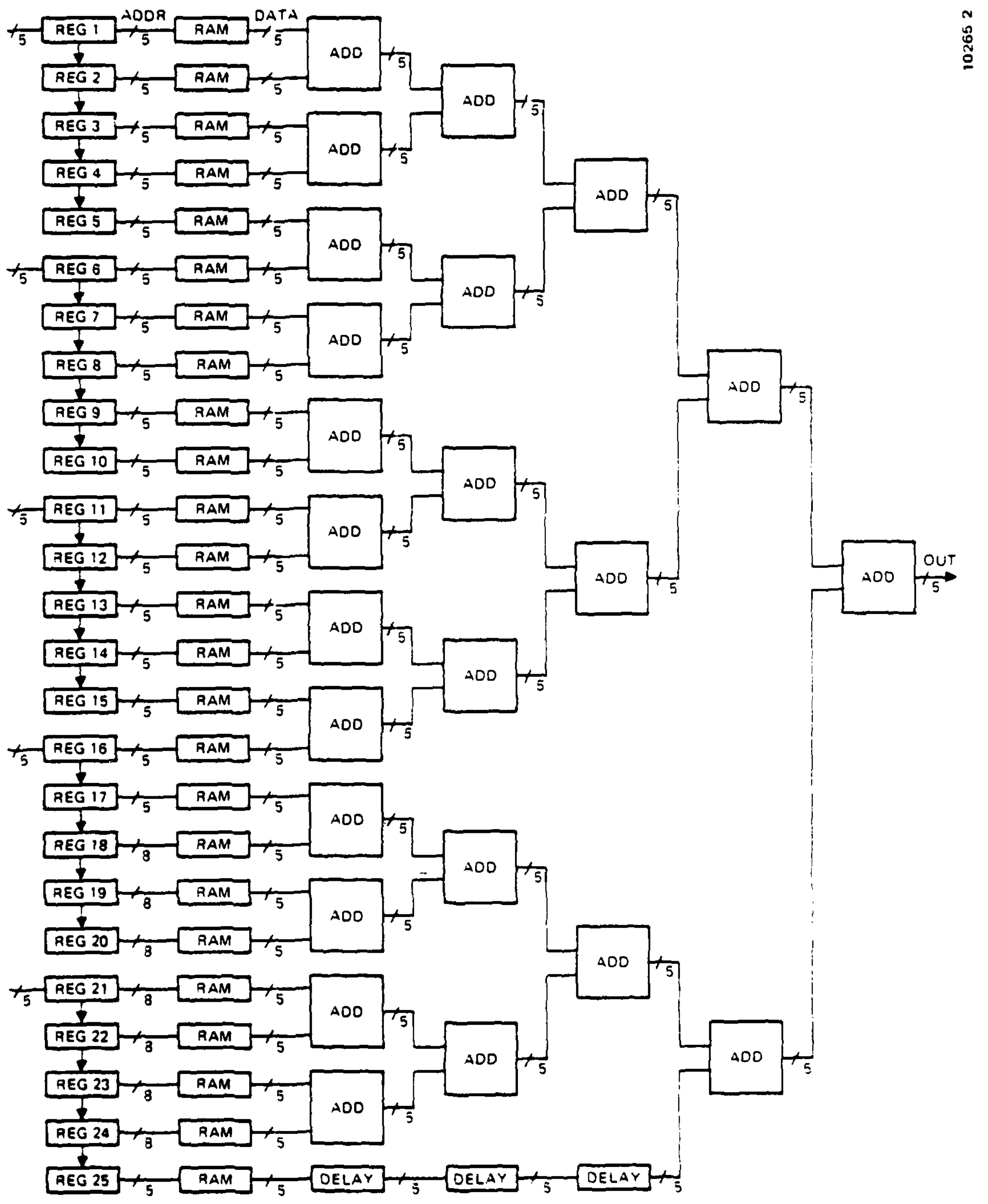


Figure 2. 5 x 5 local area processor.

base is also programmed at the same time. Lastly, concerning the programming of the RAMs, a bidirectional data bus will be provided on the data lines of the RAMs to allow programming as well as to serve as a test point in the processor. Additional control lines into a binary decoder will be required for programming and controlling the bus for testing. A possible design for the residue adder is shown in Figure 3. Essentially, the device performs a binary addition of the two operands, compares the result to the base, and subtracts the base if the result is greater than or equal to the base. The adder is programmed by providing the base value to the comparator and the two's complement value of the base to the second adder. The programming can be done in one of two ways. First, and preferable, is to provide a register on the chip for the base value and another register for the two's complement of the bases. The value in the register must then be made available to each of the residue adders that will be on the chip. The second alternative is to make the base programmable by using a ROM. This has the advantage that base values can be stored directly in the adder, which will prevent possible routing problems.

We are developing a chip based on RADIUS. However, to reduce the cost of development and the associated risk, we are restricting the chip to a 5 x 1 area processor. If this demonstration is successful, we will be able to use the designs and digitized data base as a common module to be prepared across an entire VLSI chip, resulting in a very high density, high throughput processor. Figure 4 shows the design for a 5 x 1 processor, and Figure 5 shows the configuration of a 5 x 5 convolution system which utilizes the 5 x 1 processor chip. The data are encoded using a 64 element by 20 bit ROM. The kernel is generated using four 20 bit wide line delays, 5 bits for each base. The output of the line delays is input directly to an array of 20 5 x 1 processors, five for each base. For each base, four 1024 x 5 bit ROMs are used to sum the results of the five rows. Note that conventional adders cannot be used since these additions must be done modularly, just as on the chip. Finally, 5 bits from each base convolver is input to a decoder network, which will be an array of ROMs and possibly some logic. The output of the decoder is

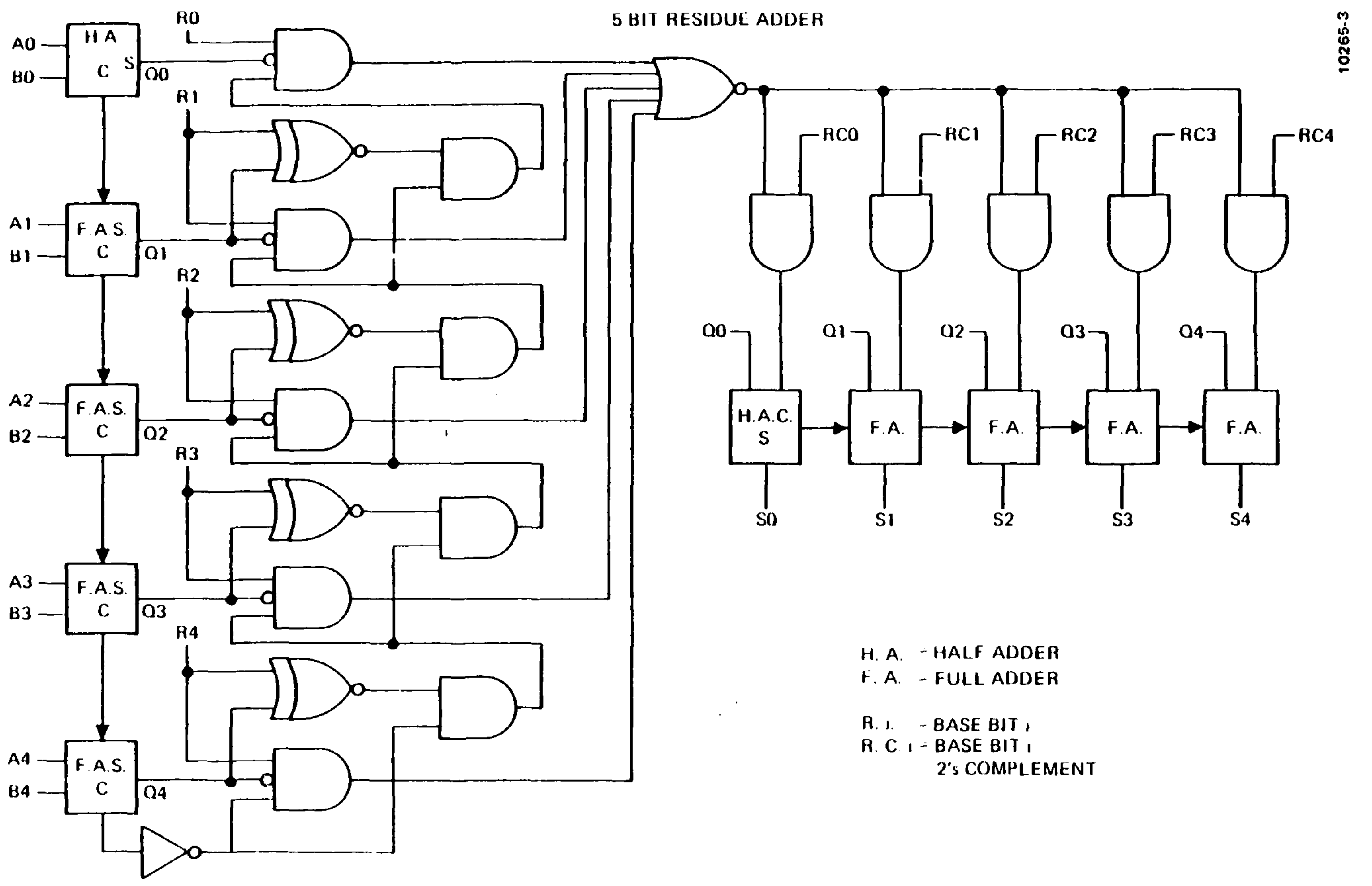


Figure 3. Residue adder.

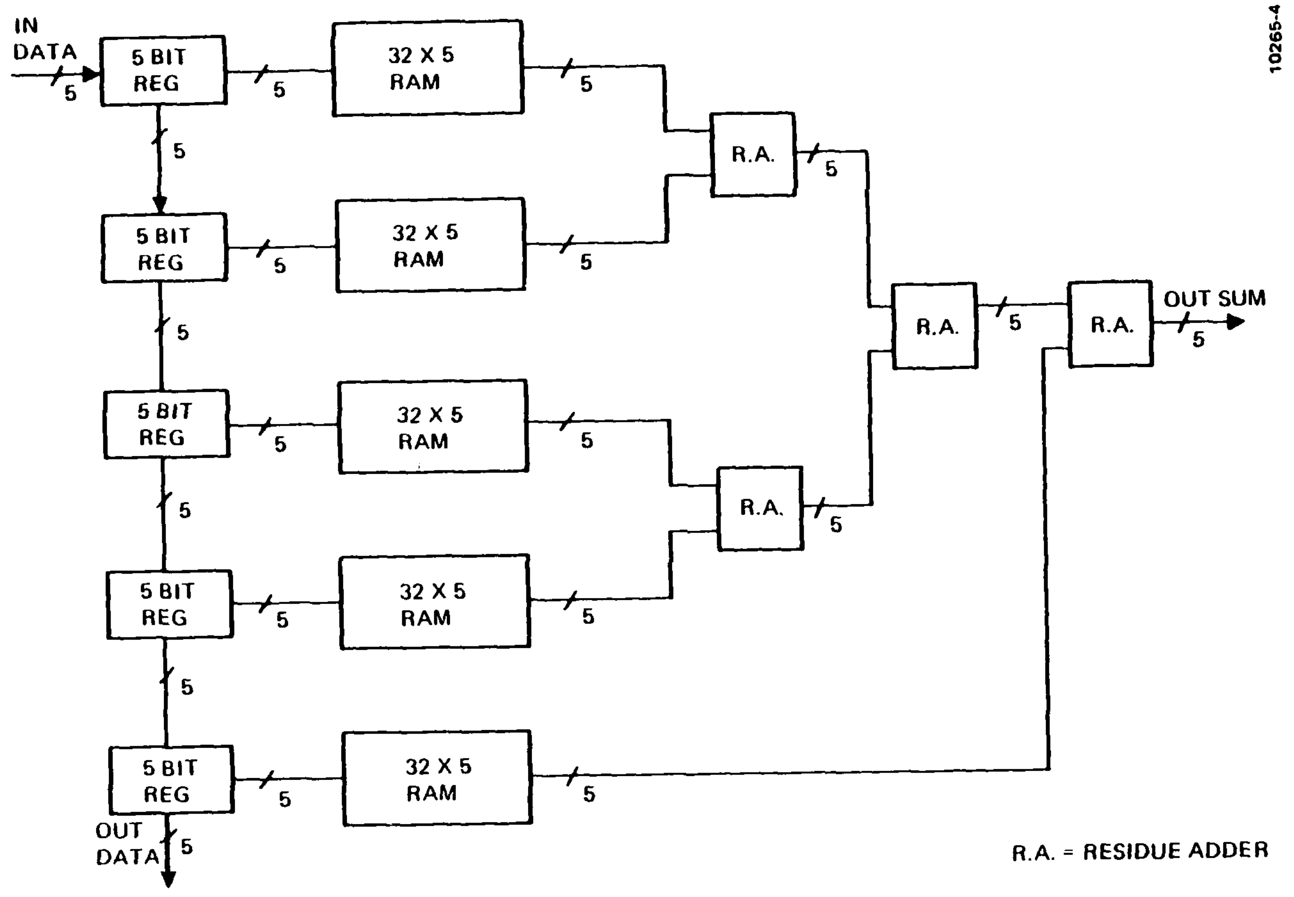


Figure 4. 5 x 1 RADIUS processor chip.

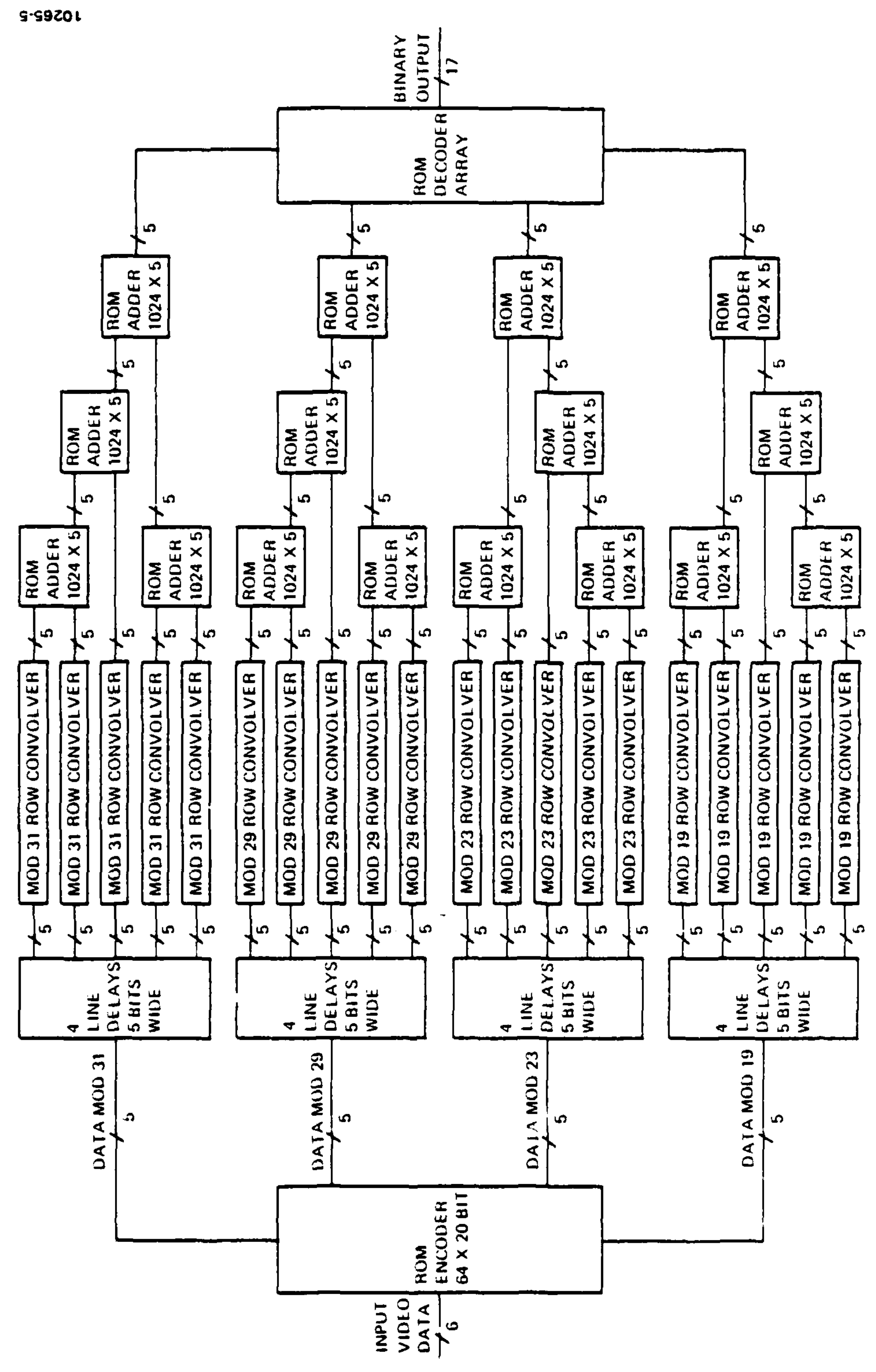


Figure 5. A 5 x 5 RADIUS processor using 5 x 1 chips.

then a 17 bit binary word. We expect that parts will be available for testing by March 1981, and that a demonstration system such as depicted in Figure 5 will be completed by July 1981.

IV. TEXTURE CLASSIFICATION SYSTEM

As mentioned earlier, logic design was performed for both the texture classification system and the line finder system. This section describes the design generated for the texture system. Figure 6 illustrates the data flow of the texture system. The input video is processed in N independent channels, where each channel generates an element of a feature vector. This means that, for each scalar pixel input, there is a vector value output, the length of the vector being equal to N, the number of channels in the system. The processing done in the channels involves a small window convolution, a normalization step, and a large window energy measure. The outputs are combined to form a vector, which is then transformed using a linear transformation. The elements of the transformed vector are used to evaluate a discriminant function, the value of which is used to perform the classification. Alternatively, the transformed feature vector can be input to a segmentation routine.

Six major functions must be performed:

- Small window convolution
- Small window statistical calculation
- Scaling
- Large window statistical calculation
- Linear transformation
- Discriminant function evaluation.

We discuss below the operations involved in each of the functions and the hardware required to perform the functions.

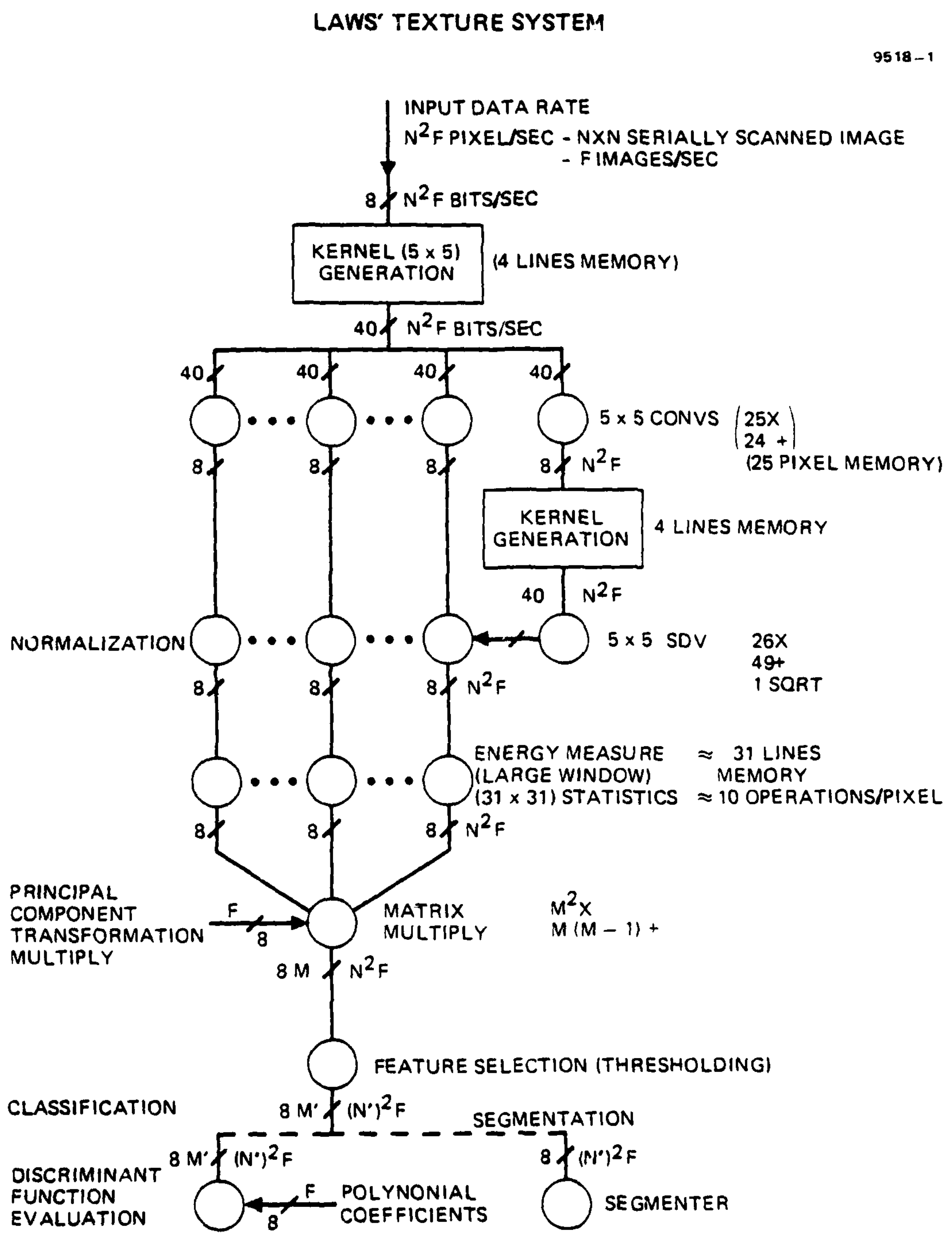


Figure 6. Laws texture classifier.

A. <u>Small Window Convolution</u>

The first step in the processing involves computing a 5 x 5 convolution. The design of a system (RADIUS) to perform this computation using the technique of residue arithmetic is presented in the previous section. The basic block that performs the 5 x 1 convolution is now being designed as an NMOS chip (CRC 181) at the Hughes Carlsbad Research Center.

B. <u>Small Window Statistical Measure</u>

A general structure for calculating a statistical moment over a two-dimensional window was described in Hughes invention disclosure PD80078; Figure 7 shows the specific structure for calculating the variance over a 5 x 5 window. This architecture assumes that the image data are being input in raster scan format. As each pixel is introduced into the system, the window that is being processed is shifted one column to the right, dropping the left-most column and adding the column on the right. This means that the function for the new window position can be calculated by subtracting the contribution of the lost column and adding the contribution from the new column. It should be noted, as illustrated in Figure 8, that as the center of the 5 x 5 window moves across the image each subsequent kernel can be formed by the removal of a single pixel at the top right, for instance, and the addition of one new pixel at the bottom left. With this proviso, we can scan the image directly by successively eliminating the 5 pixel column at the trailing edge and adding the new column on the right as shown. Figure 8 illustrates this method for updating the window function.

This technique greatly reduces the necessary data bandwidth for calculating the mean and the sum of squares for the processing window; these can then be combined to form the variance. The structure shown includes shift registers for pixel storage, column storage for the mean calculation, the column storage for the sum of squares calculation.

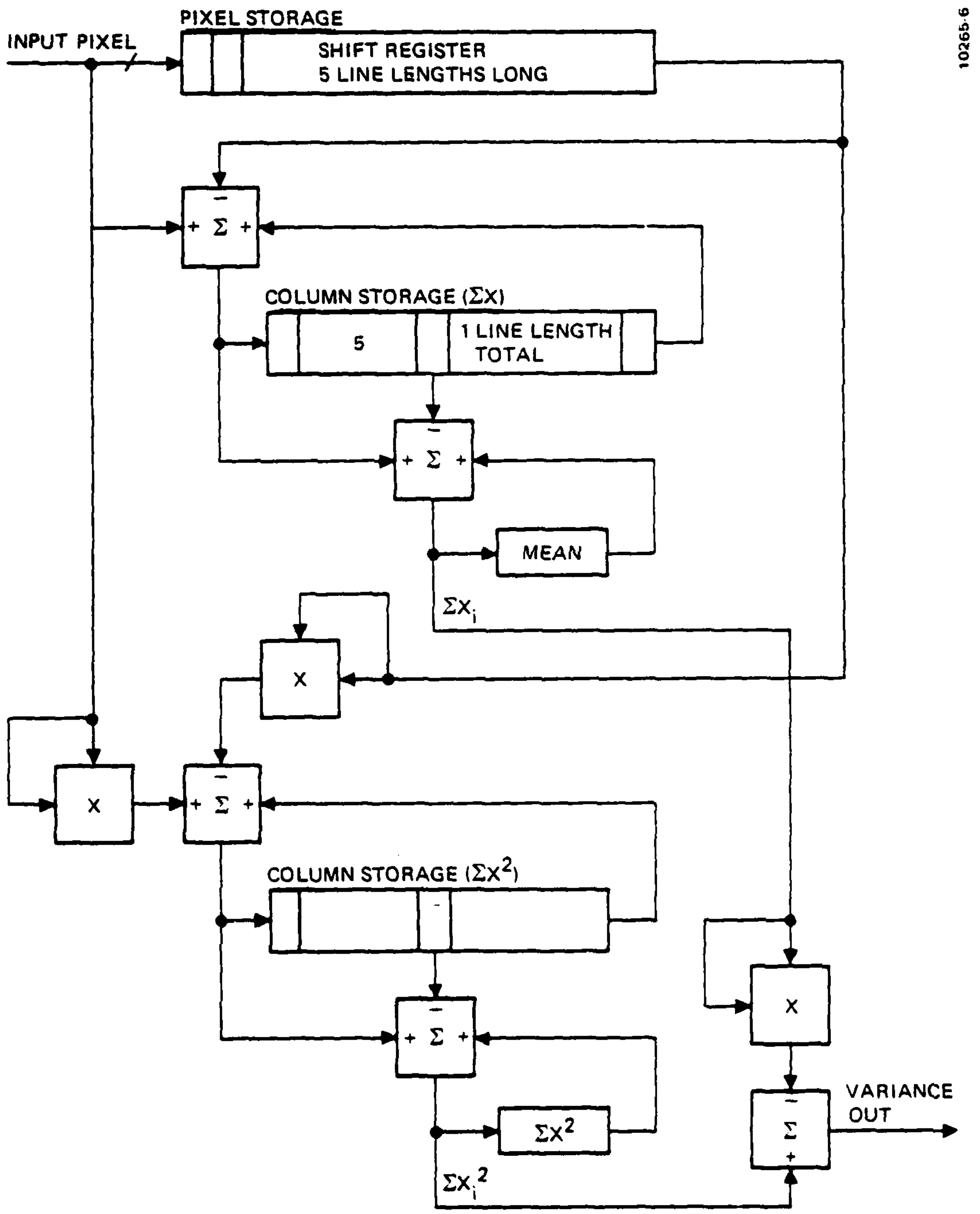


Figure 7. 5 x 5 variance calculation.

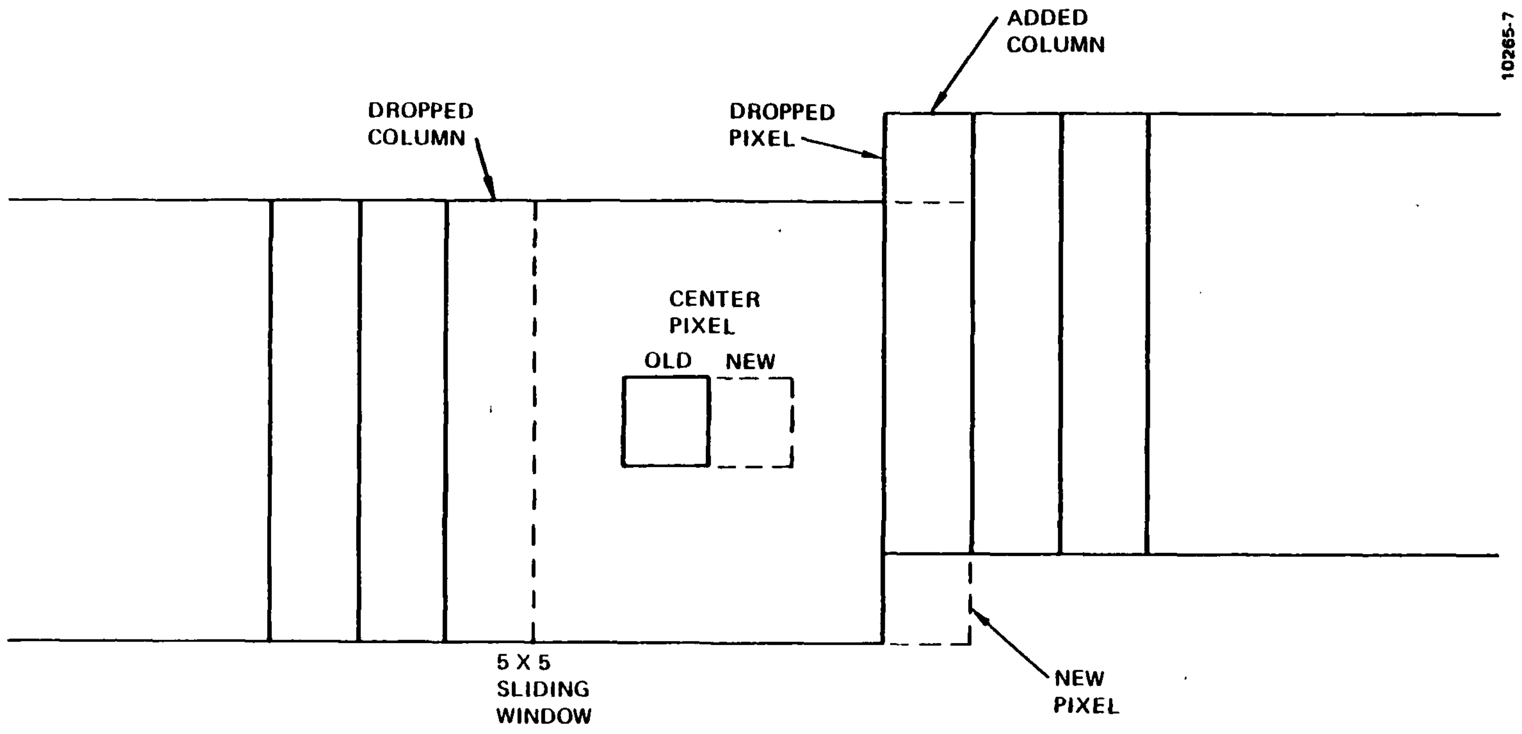


Figure 8. Sliding window updating.

The only arithmetic logic required for this structure is four 3-input adders and three multipliers.

This is a function that could be implemented using a residue technique, since only arithmetic functions are being performed. Since the preceding function is already being performed using residue arithmetic, the conversion back to binary could be done after the moment calculation. The structure would look identical, but the blocks themselves would each be smaller. For example, with a smaller wordsize the memory would be reduced. Also, the multiplier box could be replaced with look-up tables, thus providing a savings in hardware. To determine the feasibility, a statistical dynamic range analysis will be performed to see how many bases would be required and thus if the residue technique could provide normalization.

C. Normalization

The next major function to be performed is a division or nomalization function. The output of the convolutions are divided by the output of the small window moment calculation, on a pixel by pixel basis. This requires that there be some memory to delay the output of the convolutions while the moment is being calculated. The memory required would be approximately 5 line lengths x 8 bits x N channels.

A pipelined system for performing an integer divide is shown in Figure 9. This performs a division between two binary numbers using the direct method: an 8 bit divisor would require 8 stages. If more precision were required, more stages could be used, and these would calculate the fractional part of the answer. The direct method was chosen over the iterative method because of the synchronous nature of this system.

D. Large Window Statistical Calculation

The structure for performing the large window statistical calculation is the same as for the small window calculation. The function suggested by Laws is the sum of the absolute values of the pixels.

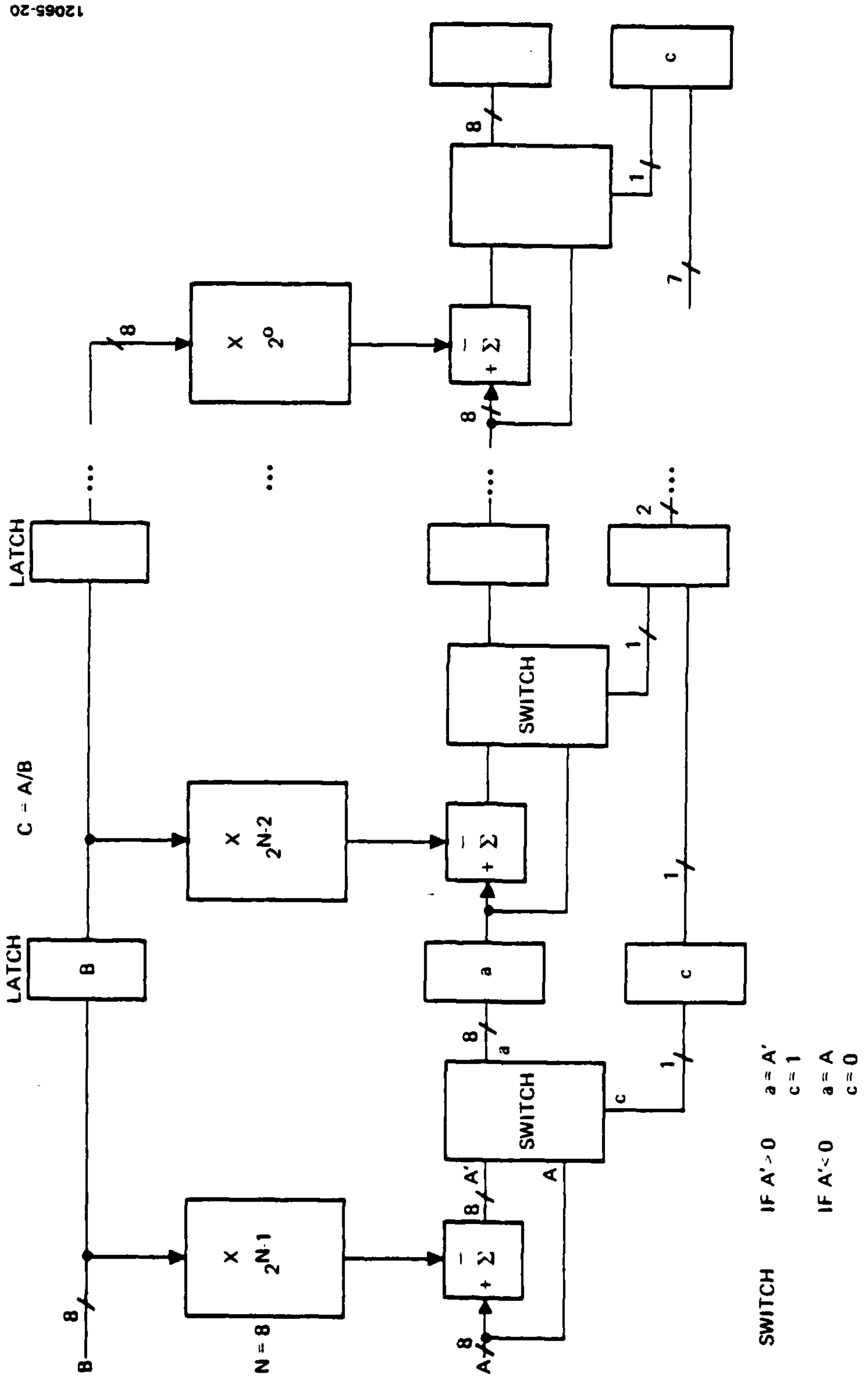


Figure 9. Structure for integer divide.

Figure 10 shows a structure for performing this function. As suggested in the discussion of the small window moment calculation, this structure could be implemented in residue. In fact, if the normalization step could be skipped, then the whole system could be accomplished in residue. There are two problems, however:

- An absolute value cannot be accomplished in residue and thus the next best thing would be to use the sum of squares.
- A preliminary dynamic range analysis using the sum of squares over a 15 x 15 window indicates that the system would need to support a dynamic range in excess of 36 bits. This would definitely be prohibitive in that a large number of bases would be required and the bases themselves would require over 5 bits to encode.

It may be that a statistical dynamic range analysis will show that it is possible to achieve a low probability of overflow with a more reasonable dynamic range (e.g., <30 bits).

E. Linear Transformation

This function will generate a vector with M components by forming linear combinations of the N components of the input vector. Since the structure shown in Figure 11 can be used to generate a single component of the output vector, the entire output vector can be generated by replicating this structure M times. This structure is basically the same as that used for the convolution. It is composed of N registers, N multipliers, and N-1 adders. As with the convolution the value of the linear weights will be programmable, with the actual mechanism for programming depending on the structure of the multiplier (either memory look-up or logic). The only difference would be in the format of the data input. For the convolution, the data are shifted through the registers; but for the linear transformation, the registers are all loaded in parallel.

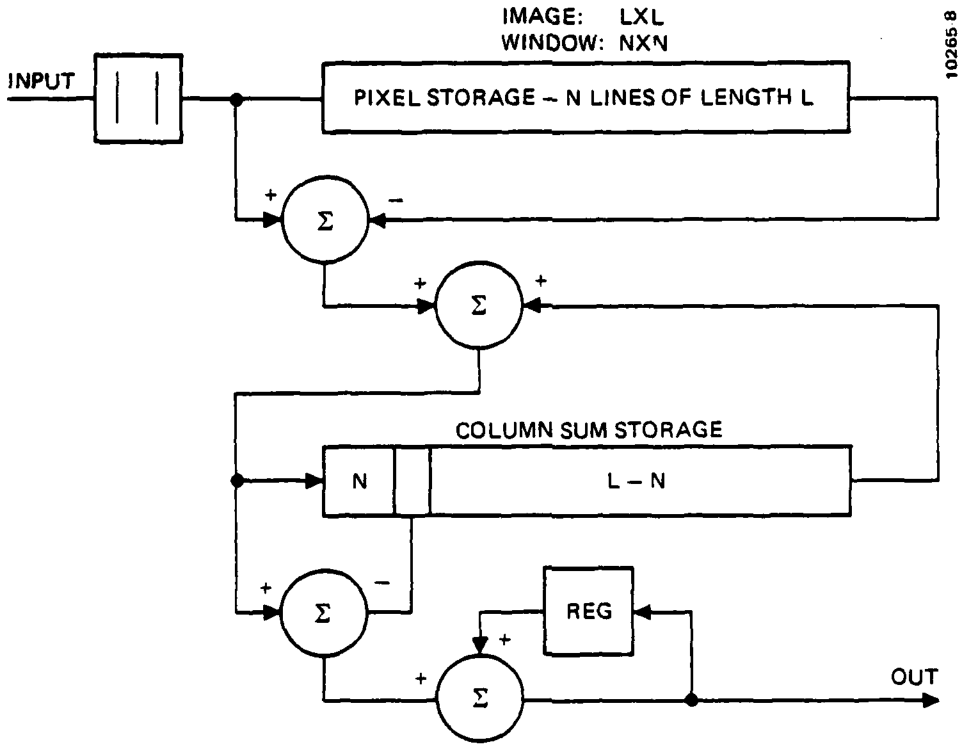


Figure 10. Large window energy measure ($\Sigma|x|$).

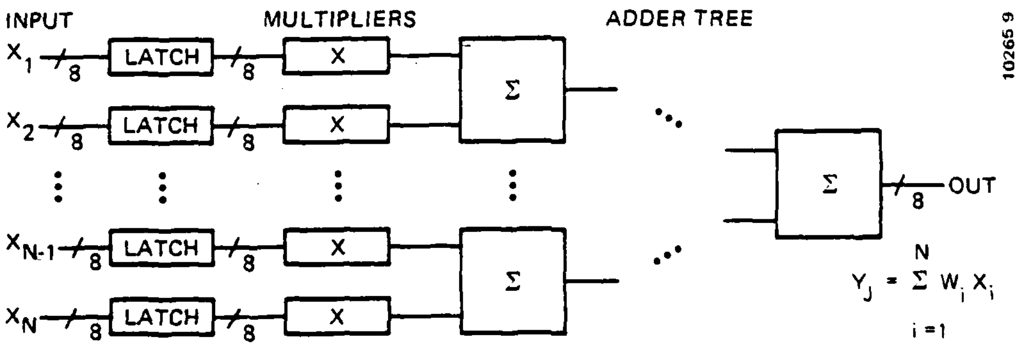


Figure 11. Linear transformation structure.

F. Discriminant Function Evaluation

No hardware was designed for this processing step since the form of the function is unknown. If the function is linear, then the structure used in the previous step could also be used for this one. Any other form, such as a higher-order polynomial, would require additonal multipliers and adders.

G. Gate Count Tabulation

Table 1 summarizes the results of the texture system design, presenting the number of gates required for each function. These data will be used later in the study for the VLSI partitioning.

V. LINE FINDER SYSTEM

This section describes the logic design of the line finder system. Figure 12 shows the data flow graph for the system and identifies four major functions:

- Edge detection
- Edge thinning
- Edge linking
- Edge tracing.

A design is presented for each of the four functions, and an estimated gate count is given.

A. Edge Strength Computation and Detection

Edge strength computation is performed by convolving 5 x 5 masks in six directions with the image of interest (Figure 13). The mask directions are set at 30° intervals, and the weights are shown in Figure 14. Following edge strength computation, the magnitudes from the six directions are compared, and the direction with greatest magnitude is selected as the edge vector for the pixel location.

There is some question as to what the optimum mask size, weights, and number of mask directions should be. For example, a larger mask size

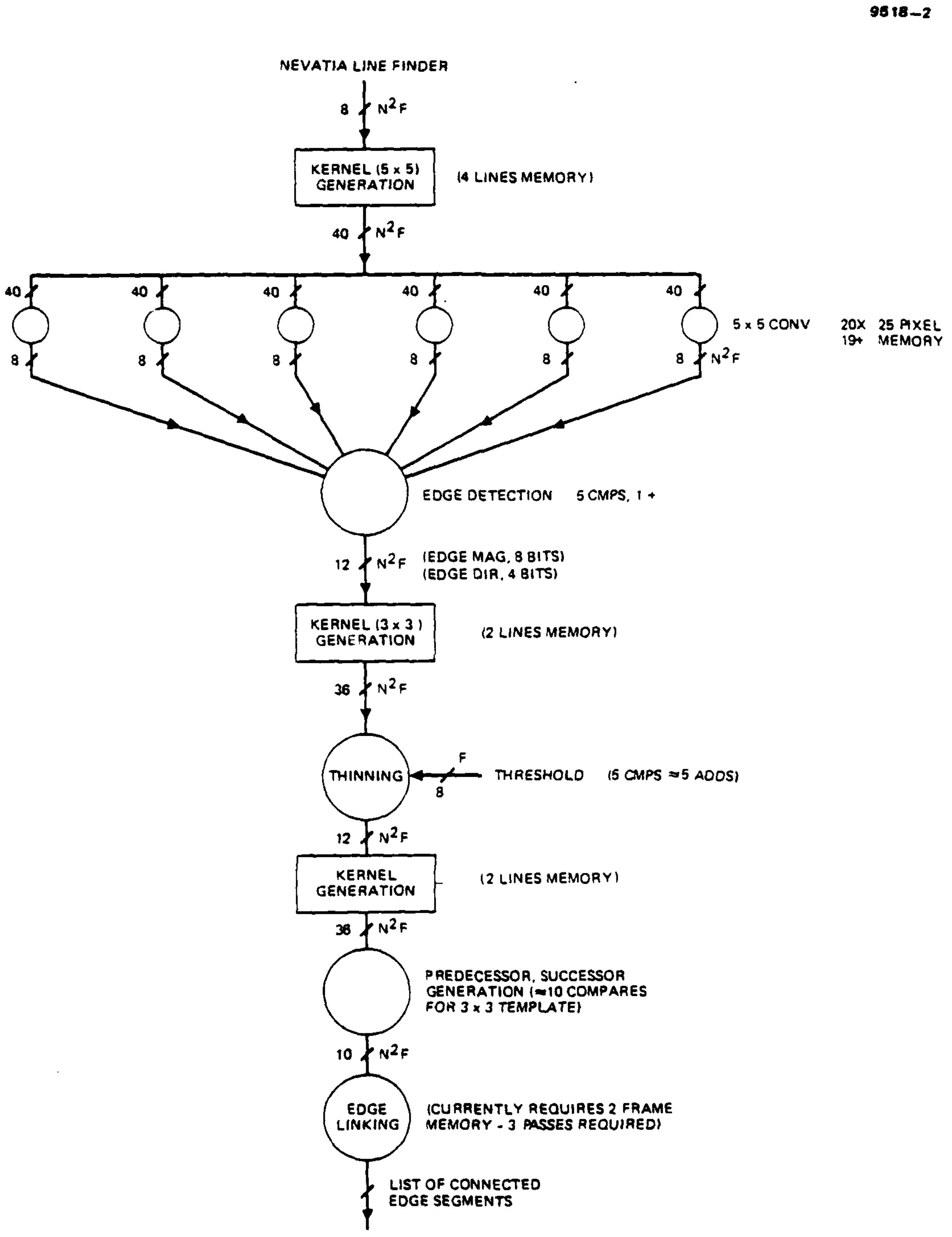

9518-2
NEVATIA LINE FINDER
8 N²F
KERNEL (5 x 5) GENERATION
(4 LINES MEMORY)
40 N²F
40
8
5 x 5 CONV
20X 25 PIXEL
19+ MEMORY
8 N²F
EDGE DETECTION 5 CMPS, 1 +
12 N²F
(EDGE MAG, 8 BITS)
(EDGE DIR, 4 BITS)
KERNEL (3 x 3) GENERATION
(2 LINES MEMORY)
36 N²F
THINNING
F
8
THRESHOLD (5 CMPS ≈5 ADDS)
12 N²F
KERNEL GENERATION
(2 LINES MEMORY)
36 N²F
PREDECESSOR, SUCCESSOR GENERATION (≈10 COMPARES FOR 3 x 3 TEMPLATE)
10 N²F
EDGE LINKING
(CURRENTLY REQUIRES 2 FRAME MEMORY - 3 PASSES REQUIRED)
LIST OF CONNECTED EDGE SEGMENTS

Figure 12. Nevatia-Babu line finder.

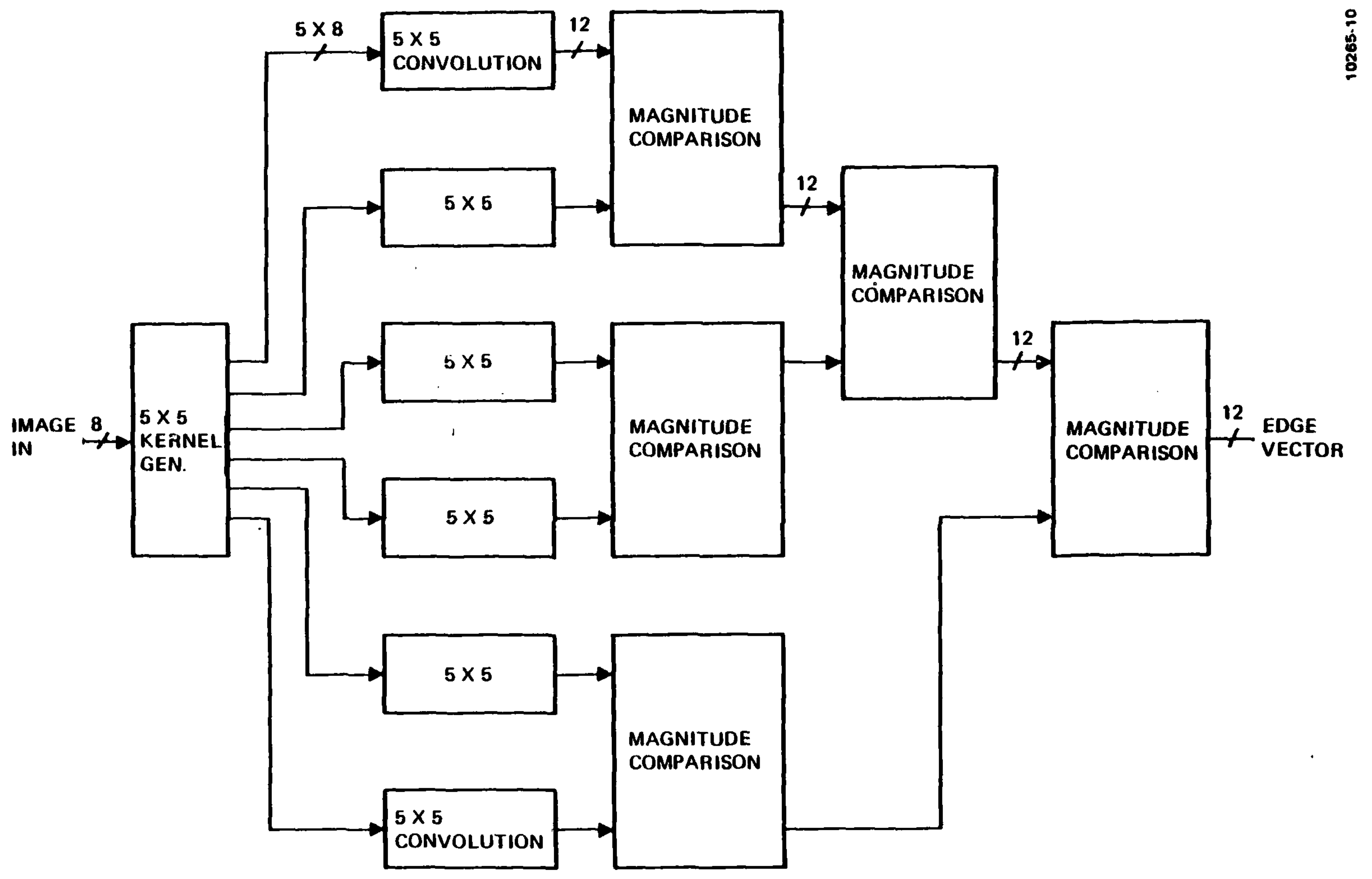


Figure 13. Logic diagram for edge detection.

10265-11

−100	−100	0	100	100
−100	−100	0	100	100
−100	−100	0	100	100
−100	−100	0	100	100
−100	−100	0	100	100

→

(a) 0°

−100	32	100	100	100
−100	−78	92	100	100
−100	−100	0	100	100
−100	−100	−92	78	100
−100	−100	−100	−32	100

↗

(b) 30°

100	100	100	100	100
−32	78	100	100	100
−100	−92	0	92	100
−100	−100	−100	−78	32
−100	−100	−100	−100	−100

↗

(c) 60°

100	100	100	100	100
100	100	100	100	100
0	0	0	0	0
−100	−100	−100	−100	−100
−100	−100	−100	−100	−100

↑

(d) 90°

100	100	100	100	100
100	100	100	78	−32
100	92	0	−92	−100
32	−78	−100	−100	−100
−100	−100	−100	−100	−100

↖

(e) 120°

100	100	100	32	−100
100	100	92	−78	−100
100	100	0	−100	−100
100	78	−92	−100	−100
100	32	−100	−100	−100

↖

(f) 150°

Figure 14. Edge masks $M_K'(i,j)$ in six directions.

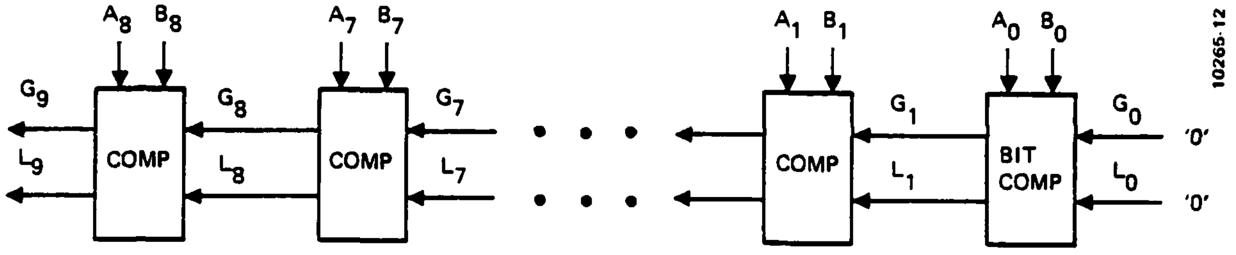
(say 5 x 5) is more immune to noise, but takes considerably more hardware to implement (than say a 3 x 3 mask). The same can be said for implementing the masks in six directions as opposed to four. It would be desirable from a hardware point of view to use a small mask size and as few mask directions as possible without sacrificing too much on performance. RADIUS can be used to perform the six convolutions. Following these six convolutions, direction tags will be added to the edge magnitudes (see Figure 13). Each data word consists of 12 bits at this stage, 8 bits for magnitude and 4 bits for direction. The magnitude part of each direction is then compared with that of the other directions, until the direction with greatest magnitude is found. Five comparators are needed to perform the magnitude comparisons.

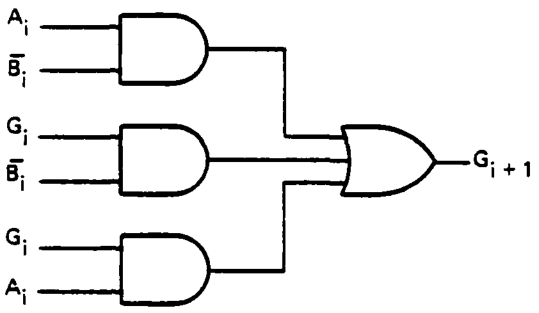
Figure 15 shows the logic for performing the magnitude comparison. The corresponding bits of each data word are compared in turn, and greater than (G) and less than (L) signals propagate towards the msb bits. G{9} and L{9} indicate whether A is greater than or less than B. If G{9} and L{9} are both low, then the two words are the same. Sixty four gates would be necessary to implement a full 8 bit comparator. Hence 320 gates would be necessary to implement the five comparators in the edge detection step. It has been estimated that it takes 178 K gates to implement the six residue convolvers. This is by var the most hardware intensive computational part.

B. Thinning and Thresholding

Thinning in Nevatia's process is accomplished by comparing the edge magnitude and direction at a pixel location with that of some of its surrounding eight neighbors. To qualify as an edge point, the following rules must be observed:

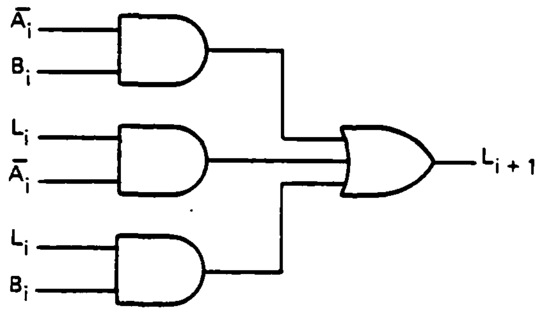
(1) The edge magnitude at the pixel location must be greater than that of its two neighbors in a direction normal to the direction of this edge. The normal to a 30° edge is approximated by the diagonals of a 3 x 3 grid.





$G_{i+1} = A_i \bar{B}_i + G_i \bar{B}_i + G_i A_i$

(4 GATES)



$L_{i+1} = \bar{A}_i B_i + L_i \bar{A}_i + L_i B_i$

(4 GATES)

TOTAL NUMBER OF GATES TO IMPLEMENT 8 BIT COMPARATOR = 8 X (4 + 4) = 64 GATES

Figure 15. Logic for performing magnitude comparison.

(2) The edge directions of the two neighboring pixels, as defined in (1), must be within 1 unit difference (30°) from that of the central pixel.

(3) The edge magnitude of the central pixel must exceed a certain fixed threshold. This threshold is arbitraritly set at some low value.

When implementing in hardware, the algorithm can be divided into two parts: The first accesses the edge magnitude and direction of the two neighbors normal to the direction of the edge. The second compares the magnitude and direction of the central pixel with those of the two neighbors and a fixed threshold to ensure that conditions (2) and (3) above are met. Figure 16 shows the logic for performing thinning and thresholding. The eight neighbors of the central pixel are first sent to a switching network, which decodes the edge direction of the central pixel and allows the edge data for the correct two neighbors to pass on to the comparison network. In the second part, the edge magnitudes and directions are compared, and, if conditions (2) and (3) are met, an edge point is presumed present. A gate count reveals that 190 gates are needed: 8 for the switching network and 181 for the comparison network.

C. Edge Linking

At this point, the data format for each pizel is pictured in Figure 17. One bit is assigned to indicate if the pixel is an edge point, 3 bits to indicate the direction of the edge, and 8 bits to indicate edge strength. The next step is to link the edge points together by forming predecessor and successor members. There are at most three possible candidates to be a predecessor or successor among the eight neighbors of an edge point; however, an edge point can have at most two predecessors and two successors. In such a case, only the primary predecessor (or successor) is encoded, and a special bit is marked to indicate the presence of a fork. The following rules are observed in edge linking:

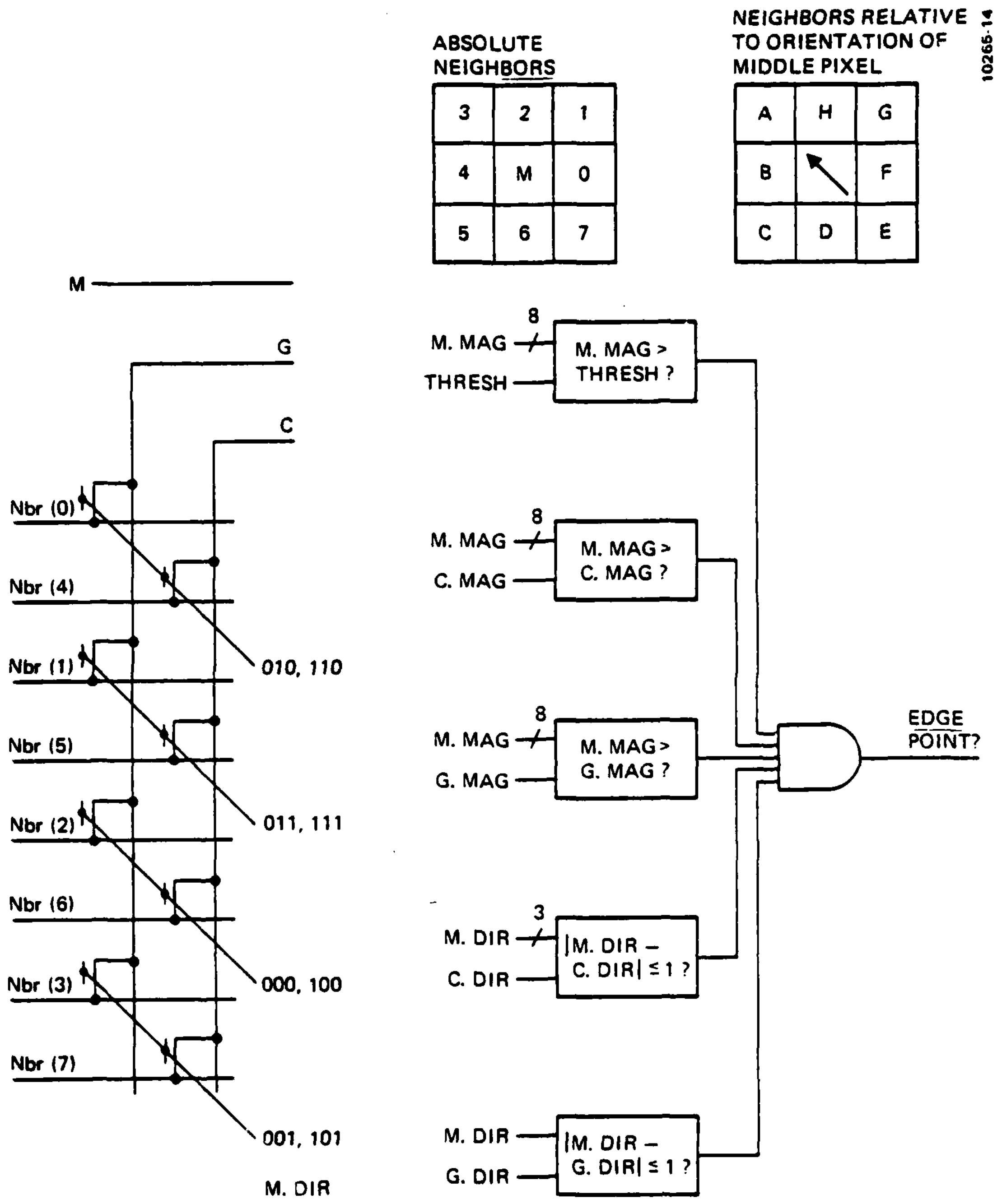


Figure 16. Logic for performing thinning and thresholding.

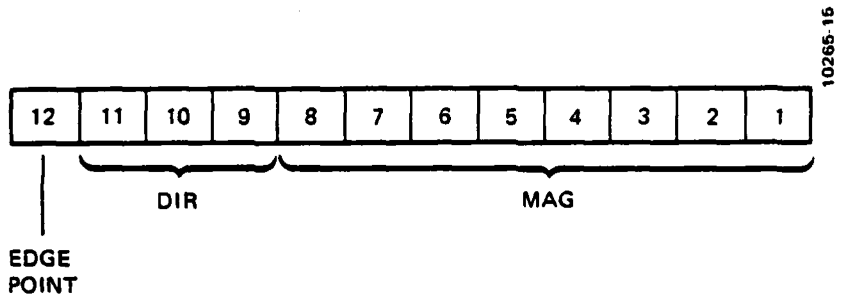


Figure 17. Data format for edge point after thinning and thresholding.

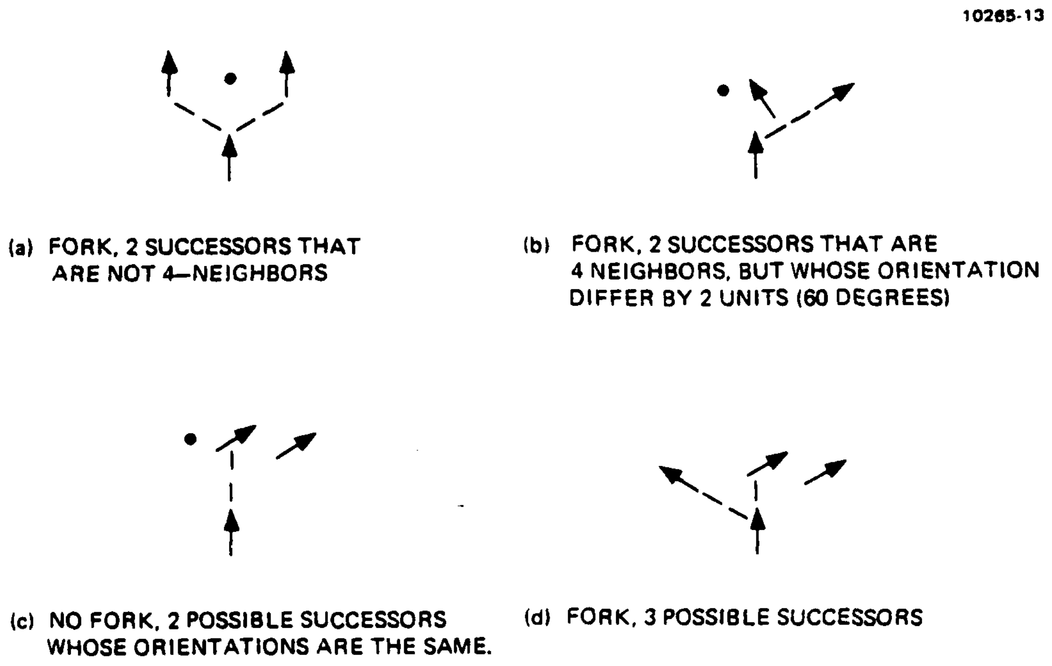

10265-13
(a) FORK, 2 SUCCESSORS THAT ARE NOT 4—NEIGHBORS
(b) FORK, 2 SUCCESSORS THAT ARE 4 NEIGHBORS, BUT WHOSE ORIENTATION DIFFER BY 2 UNITS (60 DEGREES)
(c) NO FORK, 2 POSSIBLE SUCCESSORS WHOSE ORIENTATIONS ARE THE SAME.
(d) FORK, 3 POSSIBLE SUCCESSORS

Figure 18. Some successor configurations illustrating edge linking rules.

(1) The orientation of a predecessor (successor) must be less than 1 unit difference (30°) from that of the central edge point.

(2) If there is more than 1 possible predecessor (successor), a fork will exist if they are not 4-neighbors or if their orientations differ by 2 units (60°) if they are 4-neighbors. In such a case, the predecessor (successor) with the greater magnitude is encoded as the primary predecessor (successor). If the possible candidates are 4-neighbors with the same orientation, the chosen candidate is the nearer of the two in the Euclidean sense.

The interested reader is referred to Refs. 1 and 2 for more details on the rules concerning edge linking. Figure 18 shows some successor configurations illustrating the above rules.

The logic for edge linking is shown in Figure 19. In the first section, a shifting network decodes the edge orientation of the central pixel and accesses the edge data for the three possible successor (predecessor) locations. This edge data is then sent to a comparison network that tests whether conditions (1) and (2) above have been satisfied. The results are then sent to a PLA containing about 40 minterms, the output of which indicates the successor and whether a fork is present. The coding for the PLA is shown in Figure 20. The gate count is 6 for the shifting network, 72 for the comparison network, and 170 for the PLA, for a total of about 250 gates. Formation of the predecessor numbers would take another similar sized network, bringing the total number of gates to about 500.

D. Edge Tracing

In this step, the predecessor/successor (PS) elements formed in the previous operation are linked together. The data format for the PS file is shown in Figure 21. Included in this format is a trace bit which indicates whether the point has already been collected The PS elements are linked in three passes:

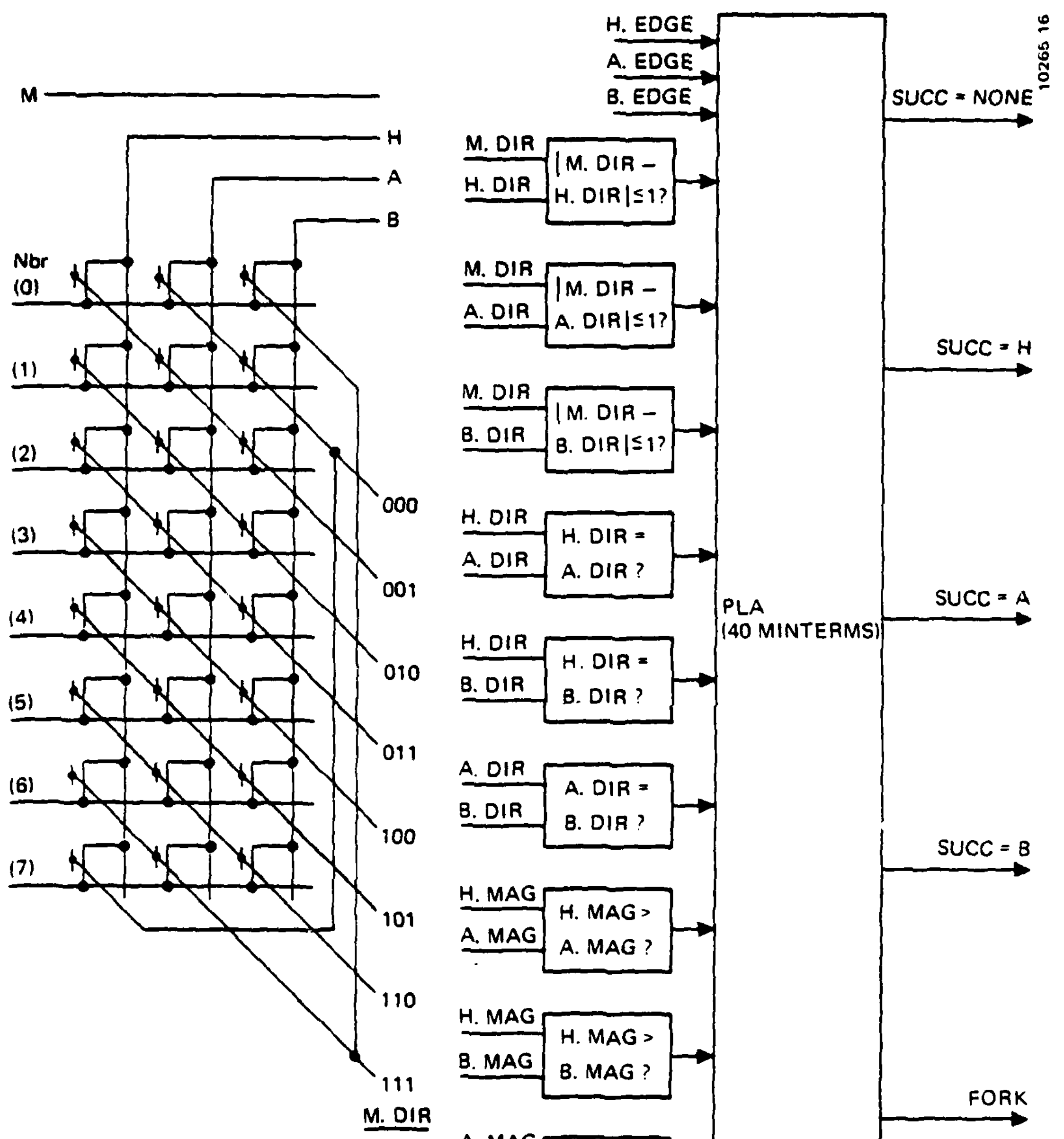


Figure 19. Logic for edge linking.

H. EDGE	A. EDGE	B. EDGE	M. DIR – H. DIR ≤ 1?	M. DIR – A. DIR ≤ 1?	M. DIR – B. DIR ≤ 1?	H. DIR = A. DIR ?	H. DIR = B. DIR ?	A. DIR = B. DIR ?	H. MAG > A. MAG ?	H. MAG > B. MAG ?	A. MAG > C. MAG ?	FORK	PRIMARY SUCC
0	0	0	–	–	–	–	–	–	–	–	–	0	0
0	0	1	–	–	0	–	–	–	–	–	–	0	0
			–	–	1	–	–	–	–	–	–	0	B
0	1	0	–	0	–	–	–	–	–	–	–	0	0
			–	1	–							0	A
1	0	0	0	–	–							0	0
			1	–	–							0	H
0	1	1	–	0	0							0	0
			–	0	1							0	B
			–	1	0							0	A
			–	1	1	–	–	0	–	–	0	1	B
									–	–	1	1	A
						–	–	1	–	–	–	0	A
1	0	1	0	–	0							0	0
			0	–	1							0	B
			1	–	0							0	H
			1	–	1	–	–	–	–	0	–	1	B
									–	1	–	1	H
1	1	0	0	0	–							0	0
			0	1	–							0	A
			1	0	–							0	H
			1	1	–	0	–	–	0	–	–	1	A
									1	–	–	1	H
						1	–	–	–	–	–	0	A
1	1	1	0	0	0							0	0
			1	0	0							0	H
			0	1	0							0	A
			0	0	1							0	B
			1	0	1	–	–	–	–	0	–	1	B
									–	1	–	1	H
			1	1	0	0	–	–	0	–	–	1	A
									1	–	–	1	H
						1	–	–	–	–	–	0	A
			0	1	1	–	–	0	–	–	0	1	B
									–	–	1	1	A
						–	–	1	–	–	–	0	A
			1	1	1	1	0	0	–	–	0	1	B
									–	–	1	1	A
						0	0	1	0	–	–	1	A
									1	–	–	1	H

10265 17

Figure 20. Coding for edge linking PLA.

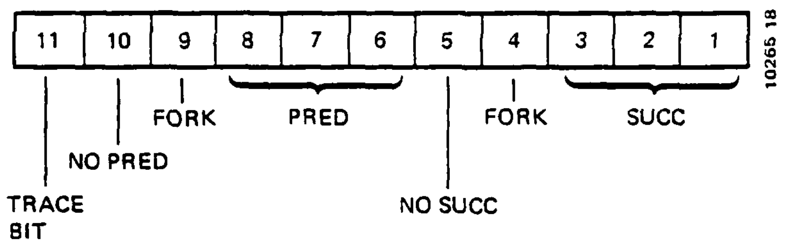


Figure 21. Predecessor/successor data format.

(1) In the first pass, a raster scan is made of the PS files in search of an edge point with no predecessor. Edge tracing begins at these points, and only the primary successor elements are linked when there is a fork.

(2) In the second pass, the edge points that have forks are revisited, and the secondary successor elements are linked.

(3) In the third pass, circular edge sements with no starting or fork points are linked. This requires scanning the trace file to find an edge point that has not been collected before and then traceing out the circular segment.

It would be undesriable to perform edge tracing in three passes if the operation is to be done in real time, since the requirements on buffering and speed of the hardware would then be quite severe. The three passes for the operation could be reduced if all the start and fork points were identified in a previous step, and the addresses stored in a list. The start address for tracing an edge segment could then be provided by this list instead of by scanning the PS file. This could result in considerable time saving, since typically less than 10% of an image are edge points and only a fraction of those are start and fork points.

The logic for edge tracing is shown in Figure 22. Two 5.5 megabit memories are needed for storing the PS files. While edge tracing is being performed on the edge data in one memory, the other memory is being filled with fresh data. The "start and fork" list provides the starting address of an edge segment. The PS information fetched for this point is used to form the edge linked list, and also to generate the address of the successor to be fetched. The address modification unit acts as a controller enforcing the three rules above and may broadcast the address of a new pixel to be fetched, modify the address of the previous pixel fetched, or decide that a fork is present and broadcast the address of the secondary successor that is to be fetched.

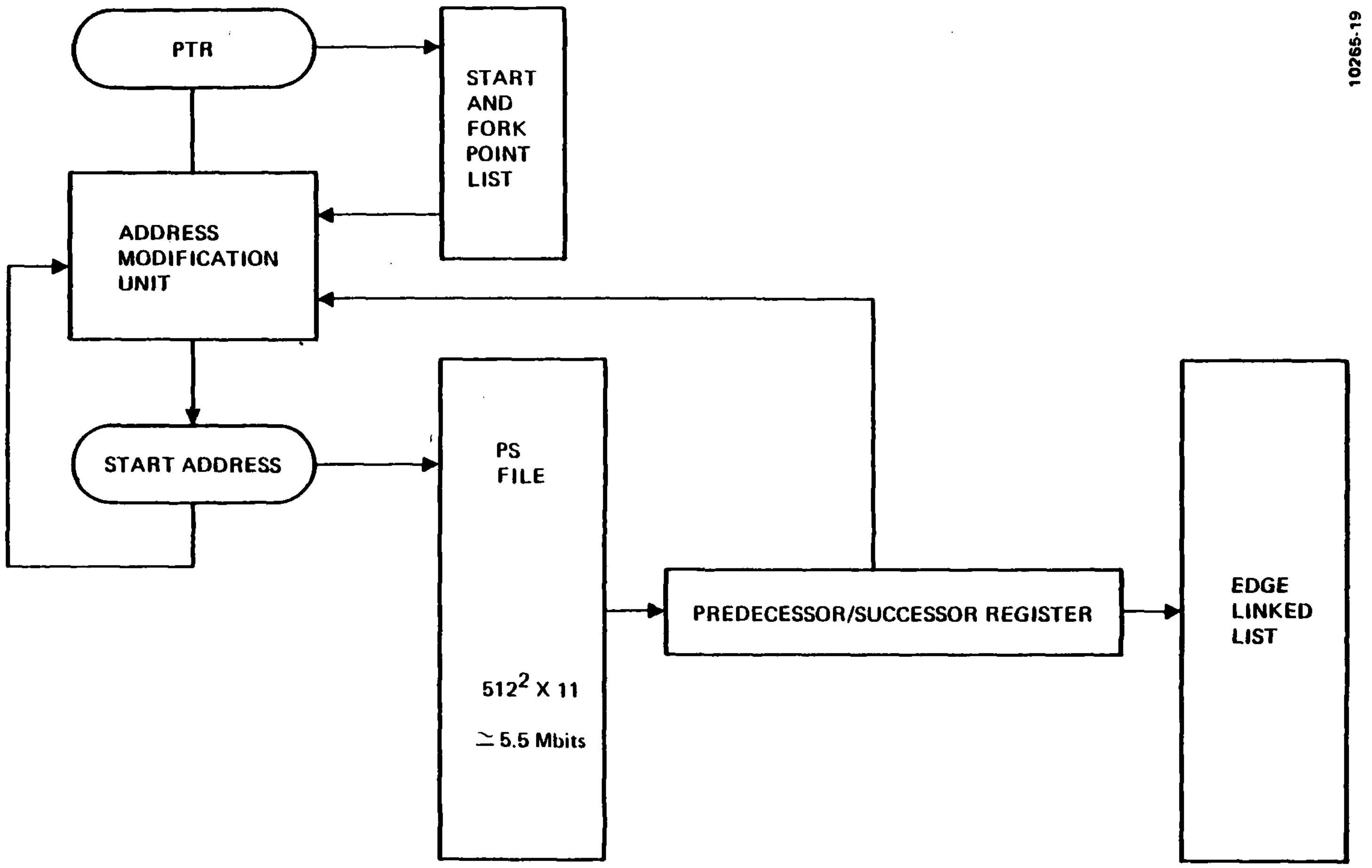


Figure 22. Logic for edge tracing.

E. Gate Count Tabulation

Table 2 summarizes the results of the line finder system, presenting the number of gates required for each function. As with the texture system, these data will be used later in the study for VLSI partitioning.

V. SYSTEM INTEGRATION TO HOST PROCESSOR

A principal area of our present work is to analyze and develop hardware that will be integrated into a commercially available test system to make it portable and extendable. One factor expected to affect the partitioning substantially is the input/output requirements of the system. The data flow graphs specify the bus widths and the basic data inputs and outputs, but absolute bandwidths and control requirements cannot be specified until an operating environment has been defined. In the past, when we have fabricated CCD LSI image processing chips, they have been used in a stand alone system with dedicated input and output devices. The bandwidths of these devices specified the bandwidth of the data paths on the chip. But we do not expect some of the systems being examined in this study to be used in a stand alone system; instead we expect them to be used as a peripheral device to a host general-purpose processor. This requires then that we expand our system study to include the implications of using a general purpose computer to host the IU systems.

We are currently acquiring a PDP 11/34 computer system to provide us with a means of collecting our own experimental data on the interface problems. Our immediate plans involve integrating the residue local area processor on a card which connects directly to the DEC UNIBUS. We plan to use direct memory access to transfer the data to and from the processor, since this will free the host processor for other duties. This experimental work will be complemented by a study of other systems that use special purpose image processing devices as peripheral devices.

VI. FUTURE WORK

The work described above will continue over the next year with particular emphasis on the following areas:

- Development of VLSI RADIUS processor
- Design and partioning of IU systems
    - Design of segmenter system
    - Commonality studies
- Simulations cf designs
- Integration of system to host PDP 11-34.

REFERENCES

1. R. Nevatia and K.R. Babu, "Linear Feature Extraction and Description," to be published in CUIP, 1980.

2. R. Nevatia and K.R. Babu, "An Edge Detection, Linking and Line Finding Program," USC IPI Report No. 840, Sept. 1978.

3. C.S. Swigert, "Decoding the Edge Detection Linking and Line Finding Program of R. Nevatia and R. Babu," HAC Memo ESD 545, Oct. 1979.

4. V.S. Wong, "Description of Edge Finding Process in Thesis by V.S. Wong," HAC Memo ESD 303, Aug. 1980.

5. S.D. Fouse, "Proposal for Residue Convolver Chip," HAC Memo ESD 242, July 1980.

6. R. Babu, Private communication.

7. R.O. Duda and P.E. Hart, "Pattern Classification and Scene Analysis," Wiley, 1973, pp. 338-339.

Table 1. Gate Count for Texture Classification System

5 line kernel generation	15 K gates
5 x 5 convolution	27K gates/channel
5 x 5 variance	10K gates
Normalization	1K/channel
Large window statistical calculation	8.25K/channel
Transform (M input channels)	2.1K·M/output channel

Table 2. Gate Count for Line Finder System

Edge detection	178K
Thinning	190 gates
Edge linking	500
Edge tracing	12 Mbit memory + 5K logic gates

4. RECENT INSTITUTE PERSONNEL PUBLICATIONS AND PRESENTATIONS

1. B. Bhanu and O.D. Faugeras, "Segmentation of Images Having Unimodal Distributions," Submitted to the IEEE Trans. on Pattern Analysis and Machine Intelligence, July 1980.

2. B. Bhanu and J.H. McClellan, "On the Computation of the Complex Cepstrum," IEEE Trans. on Acoustics, Speecn and Signal Processing, pp. 108-111, October 1980.

3. P. Chavel, A.A. Sawchuk, T.C. Strand, A.R. Tanguay, Jr., B.H. Soffer, "Optical Logic with Variable Grating Mode Liquid-Crystal Devices," Optics Letters, Vol. 5, pp. 398-400, (September 1980).

4. O.D. Faugeras, "An Optimization Approach for Using Contextual Information in Computer Vision," 1st Annual Conference on Artificial Intelligence, Stanford, August 1980.

5. O.D. Faugeras, "Autoregressive Modeling with Conditional Expectations for Texture Synthesis," 5th International Conference on Pattern Recognition, 1980.

6. O.D. Faugeras, "Decomposition and Decentralization Techniques in Relaxation Labeling," to be published in Computer Graphics and Image Processing, 1980.

7. O.D. Faugeras, "Decomposition and Decentralization Techniques in Relaxation Labeling," 1st European Conference on Signal Processing (EUSIPCO-80), Lausanne, September 1980.

8. O.D. Faugeras, "Des chiffres et des Images," Special Issue of

Science et Avenu, No. 29, pp. 58-63, 1980 (invited paper, in French).

9. O.D. Faugeras, "Optimization Techniques in Scene Analysis," 4th International Conference on Analysis and Optimization, Versailles, France, December 16-19, 1980.

10. O.D. Faugeras and J.F. Abramatic, "Design of 2-D Filters from Small Generating Kernels," submitted to the IEEE Trans. on Acoustics, Speech and Signal Processing, September 1980.

11. O.D. Faugeras and D.D. Garber, "Algebraic Reconstruction Techniques for Texture Synthesis," 5th International Conference on Pattern Recognition, Miami Beach, December 1980.

12. O.D. Faugeras and W.K. Pratt, "Decorrelation Methods of Texture Feature Extraction," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 4, July 1980.

13. O.D. Faugeras, W.K. Pratt, and A. Gagalowicz, "Applications of Stochastic Texture Field Models to Image Processing," to be published in a Special Issue of the IEEE Proceedings, 1981.

14. O.D. Faugeras and K.E. Price, "Matching Symbolic Descriptions and Application to the Semantic Description of Aerial Photographs," submitted to the IEEE Trans. on Pattern Analysis and Machine Intelligence, June 1980.

15. O.D. Faugeras and K.E. Price, "Semantic Description of Aerial Images Using Stochastic Labeling," Proceedings of the IU Workshop, University of Maryland, pp. 89-94, April 1980.

16. O.D. Faugeras and K.E. Price, "Semantic Description of Aerial Images Using Stochastic Labeling," 5th International Conference on Pattern Recognition, Miami Beach, December 1980.

17. S. Inokuchi and R. Nevatia, "Boundary Detection in Range Pictures," to be presented at International Conference on Pattern Recognition, Miami Beach, December 1980.

18. C.M. Lo and A.A. Sawchuk, "Restoration with Poisson Noise," Proceedings of the IEEE, Vol 68, 1980.

19. J. Mantock, A.A. Sawchuk and T.C. Strand, "Hybrid Optical/Digital Texture Analysis," Optical Engineering, Vol. 19, March/April 1980.

20. R. Nevatia, "Image Understanding Research at USC," Seminar presented at Univ. of California, Irvine, March 1980.

21. R. Nevatia, Panel Discussion on Image Processing, NCC. 1980, May 1980, Anaheim, California.

22. R. Nevatia and K.R. Babu, "Linear Feature Extraction and Description," Computer Graphics and Image Processing, July 1980, pp. 257-269.

23. R. Nevatia, with C. Clark et al., "Matching of Natural Terrain Scenes," to be presented at International Conference on Pattern Recognition, Miami Beach, December 1980.

24. A.A. Sawchuk, J. Bescos, and I. Glaser, "Restoration of Color Images Degraded by Chromatic Aberrations," Applied Optics, Vol. 19, November 1980.

25. A.A. Sawchuk, T.C. Strand, P. Chavel, "Polychromatic Optical Processing," Gordon Research Conference on Coherent Optics and Holography, Santa Barbara, California, June 1980.

26. T.C. Strand, A.A. Sawchuk, A.R. Tanguay, Jr., P. Chavel,

D. Boswell, A.M. Lackner and B.H. Soffer, "Variable Grating Mode Liquid Crystal Valves with their Application to Optical Processing," Gordon Research Conference on Coherent Optics and Holography, Santa Barbara, California, June 1980.

27. F. Vilnrotter, R. Nevatia and K.E. Price, "Structural Description of Natural Textures," to be presented at International Conference on Pattern Recognition, Miami Beach, December 1980.